An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau Ben Moszkowski David Smallwood

October 26, 2026

Abstract

These Isabelle theories introduce the semantics and syntax of Finite and Infinite Interval Temporal Logic (ITL). The ITL proof system, as introduced in [11, 15], has been encoded and its soundness has been checked. The encoding is shallow using the Intensional Logic technique of [9]. An extensive library of Finite and Infinite ITL theorems, taken from [14], has been checked. Non-empty coinductive lists are used to denote intervals.

Contents

1 Model: Intervals **7**

1.1 Extra operations on coinductive lists . . . 7
1.1.1 Auxiliary lemmas . . . 8
1.1.2 lbutlast . . . 9
1.1.3 lfuse . . . 15
1.1.4 ridx and lidx . . . 22
1.1.5 lsub . . . 36
1.1.6 llastlfirst . . . 42
1.1.7 lfusecat . . . 42
1.1.8 kfilter . . . 80
1.1.9 lrev . . . 114
1.1.10 Transfer rules . . . 119
1.2 Non-empty coinductive lists . . . 120
1.2.1 Type definition . . . 120
1.2.2 Code generator setup . . . 121
1.2.3 Connection with *'a llist* . . . 123
1.2.4 The nlast element *nlast* . . . 136
1.2.5 *nset* . . . 137
1.2.6 *nmap* . . . 139
1.2.7 Appending two nonempty lazy lists *nappend* . . . 140
1.2.8 Appending a nonempty lazy list to a lazy list *lappendn* . . . 142
1.2.9 The length of a nonempty lazy list *nlength* . . . 142
1.2.10 The nth element of a nonempty lazy list *nnth* . . . 143
1.2.11 *ntake* . . . 145
1.2.12 *ntaken* . . . 149
1.2.13 Concatenating a nonempty lazy list of nonempty lazy lists *nconcat* . . . 153
1.2.14 *nellist-all2* . . . 166
1.2.15 From a nonempty lazy list to a lazy list *llist-of-nellist* . . . 172
1.2.16 *ndropn* . . . 173
1.2.17 *nzip* . . . 180
1.2.18 *niterates* . . . 185
1.2.19 Filtering non-empty lazy lists *nfilter* . . . 187
1.2.20 Setup for Lifting/Transfer . . . 197
1.3 Extra operations on non-empty conductive lists . . . 201
1.3.1 ndropns . . . 201
1.3.2 lsum . . . 204
1.3.3 Definitions . . . 204

1.3.4 nbutlast . . . 205
1.3.5 nsubn . . . 208
1.3.6 nfuse . . . 213
1.3.7 nridx and nidx . . . 219
1.3.8 nlastnfirst . . . 235
1.3.9 nfusecat . . . 238
1.3.10 nkfilter . . . 252
1.3.11 nrev . . . 294
1.3.12 Lsum . . . 300
1.3.13 nsubn_nellist . . . 316
1.3.14 Transfer rule . . . 352
1.4 Extended Integers . . . 353
1.4.1 Distributive lattice . . . 377
1.4.2 Extended int intervals . . . 379
1.4.3 Relation to *enat* . . . 382

2 Update **385**
2.1 Intervals . . . 385
2.2 Semantics Primitive Operators . . . 385
2.2.1 Definitions . . . 385
2.2.2 Lemmas . . . 386

3 Shallow embedding of Finite and Infinite ITL **428**
3.1 Semantics . . . 428
3.1.1 Types of Formulas . . . 428
3.1.2 Semantics of ITL . . . 429
3.1.3 Abbreviations . . . 430
3.1.4 Properties of Operators . . . 437
3.1.5 Soundness Axioms . . . 446
3.1.6 Quantification over State (Flexible) Variables . . . 452
3.1.7 Temporal Quantifiers . . . 452
3.1.8 bind . . . 452
3.2 Axioms and Rules . . . 453
3.2.1 Rules . . . 453
3.2.2 Axioms . . . 454
3.2.3 Additional Lemmas . . . 454
3.2.4 Quantification . . . 455
3.2.5 Lemmas about *current-val* . . . 456
3.2.6 Lemmas about *next-val* . . . 456
3.2.7 Lemmas about *fin-val* . . . 457
3.2.8 Lemmas about *penult-val* . . . 457
3.2.9 Basic temporal variables properties . . . 458
3.3 Weak chop operator . . . 458
3.3.1 Propositional reasoning . . . 459
3.3.2 State formulas . . . 460
3.3.3 finite and inf properties . . . 461
3.3.4 Basic Theorems . . . 464

3.3.5 Further Properties Di and Bi . . . 477
3.3.6 Properties of Da and Ba . . . 483
3.3.7 Properties of Fin . . . 491
3.3.8 Properties of Halt . . . 518
3.3.9 Properties of Groups of chops . . . 524
3.4 Strong chop operator . . . 524
3.4.1 Strong Chop axioms . . . 524
3.4.2 ITL operators in terms of SChop . . . 526
3.4.3 Basic Theorems . . . 526
3.4.4 Further Properties Df and Bf . . . 532
3.4.5 Properties of SDa and SBa . . . 540
3.4.6 Properties of SFin . . . 550
3.4.7 Properties of Halt . . . 576
3.4.8 Properties of Groups of strong chops . . . 581
3.5 Chopstar and variants . . . 581
3.5.1 Definitions . . . 581
3.5.2 Semantic lemmas . . . 585
3.5.3 Helper lemmas . . . 602
3.5.4 Properties of Chopstar and Chopplus . . . 602
3.5.5 Kleene Algebra . . . 609
3.5.6 WPowerstar . . . 652
3.5.7 Len . . . 666
3.5.8 Properties of While . . . 670
3.6 Omega and variants . . . 686
3.6.1 Definitions . . . 686
3.6.2 Omega algebra . . . 707
3.6.3 Properties of Omega . . . 729
3.7 Projection operator . . . 740
3.7.1 Definitions . . . 740
3.7.2 Lemmas . . . 742
3.7.3 Soundness of Projection Axioms . . . 811
3.7.4 Axioms . . . 855
3.7.5 Theorems . . . 857
3.8 Until operator . . . 879
3.8.1 Definitions . . . 879
3.8.2 Axioms . . . 880
3.8.3 Theorems . . . 891
3.9 Pi operator . . . 917
3.9.1 Definitions . . . 917
3.9.2 Semantic Lemmas . . . 918
3.9.3 Soundness of Axioms . . . 924
3.9.4 Theorems . . . 956
3.9.5 SChopstar and Pi . . . 970
3.9.6 Omega and Pi . . . 981
3.10 Quantifiers . . . 989
3.11 Bind operator . . . 995
3.12 Time reversal operator . . . 995

3.12.1 Definition . . . 996
3.12.2 Finite theorems . . . 996
3.12.3 Time reversal Rules . . . 999
3.12.4 Properties of Time Reversal . . . 1009
3.13 First occurrence operator . . . 1033
3.13.1 Definitions . . . 1033
3.13.2 Semantic Theorems . . . 1035
3.13.3 Bi induction . . . 1035
3.13.4 Theorems . . . 1040
3.14 Monitors . . . 1109
3.14.1 Syntax . . . 1110
3.14.2 Derived Monitors . . . 1111
3.14.3 Monitor Laws . . . 1114
3.15 Monitor Example . . . 1136
3.16 Interval Temporal Algebra . . . 1142
3.16.1 Definition of Set of intervals and Operations on them . . . 1142
3.16.2 Simplification Lemmas . . . 1144
3.16.3 Algebraic Laws . . . 1149
3.16.4 Derived Laws . . . 1168
3.16.5 Extra Laws . . . 1176
3.16.6 Link between Set of Intervals and ITL . . . 1210

4 Deep embedding of PITL 1218
4.1 Syntax . . . 1218
4.1.1 Primitive formulae . . . 1218
4.1.2 state formula . . . 1218
4.1.3 Derived Boolean Operators . . . 1218
4.1.4 more, empty, skip and wnext . . . 1219
4.1.5 ifinite and inf . . . 1219
4.1.6 weak chop . . . 1219
4.1.7 Box and Diamond Operators . . . 1220
4.1.8 Initial and Final Operators . . . 1220
4.1.9 Big operations . . . 1220
4.2 Semantics . . . 1220
4.2.1 Intervals . . . 1220
4.2.2 Semantics Primitive Operators . . . 1220
4.2.3 Validity . . . 1225
4.3 Guarded Normal Form . . . 1237
4.3.1 Definitions . . . 1237
4.3.2 Mutex lemmas . . . 1240
4.3.3 Add to lemmas . . . 1242
4.3.4 GNF lemmas . . . 1258

5 Deep embedding of QPITL 1290
5.1 Syntax . . . 1290
5.1.1 Primitive formulae . . . 1290
5.1.2 state formula . . . 1290

5.1.3 Derived Boolean Operators . . . 1291
5.1.4 more, empty, skip and wnext . . . 1291
5.1.5 ifinite and inf . . . 1291
5.1.6 weak chop . . . 1292
5.1.7 Box and Diamond Operators . . . 1292
5.1.8 Initial and Final Operators . . . 1292
5.1.9 Forall . . . 1293
5.1.10 bind . . . 1293
5.1.11 Big operations . . . 1293
5.2 Semantics . . . 1293
5.2.1 Semantics Primitive Operators . . . 1293
5.2.2 Validity . . . 1325
5.2.3 Quantifier lemmas . . . 1336
5.3 Guarded Normal Form . . . 1346
5.3.1 Definitions . . . 1346
5.3.2 Mutex lemmas . . . 1349
5.3.3 Add to lemmas . . . 1351
5.3.4 GNF lemmas . . . 1367

6 CDT-PITL 1403
6.1 Syntax . . . 1403
6.1.1 Primitive formulae . . . 1403
6.1.2 Derived Boolean Operators . . . 1403
6.1.3 ifinite and inf . . . 1404
6.1.4 weak chop . . . 1404
6.1.5 more, skip, next, wnext, prev and wprev . . . 1404
6.1.6 Box and Diamond Operators . . . 1404
6.1.7 Initial and Final Operators . . . 1405
6.1.8 Horizontal . . . 1405
6.1.9 Compass operators . . . 1405
6.2 Semantics . . . 1407
6.2.1 Intervals . . . 1407
6.2.2 Validity . . . 1421

7 NL-ITL 1433
7.1 Syntax . . . 1433
7.1.1 Primitive formulae . . . 1433
7.1.2 state, introspective, future and past formula . . . 1434
7.1.3 Derived Operators . . . 1435
7.1.4 more, empty, skip and wnext . . . 1436
7.1.5 rfinite, lfinite, rinf and linf . . . 1437
7.1.6 weak chop . . . 1438
7.1.7 Box and Diamond Operators . . . 1438
7.1.8 Initial and Final Operators . . . 1439
7.1.9 Forall . . . 1439
7.1.10 Big operations . . . 1439
7.2 Semantics . . . 1440

7.2.1 Intervals . . . 1440
7.2.2 Semantics Primitive Operators . . . 1440
7.3 Validity . . . 1476
7.4 Deep embedding of Fusion Logic with past operators [3]. . . . 1504
7.5 Synta and Semanticsx . . . 1505
7.5.1 Primitive formulae . . . 1505
7.5.2 Intervals . . . 1505
7.5.3 Derived Boolean Operators . . . 1507
7.5.4 Future fusion expressions . . . 1508
7.5.5 Past fusion expressions . . . 1509
7.5.6 Next and Previous . . . 1509
7.5.7 Box and Diamond Operators . . . 1509
7.5.8 more, empty, skip and wnext . . . 1510
7.5.9 pmore, pempty, pskip and wprev . . . 1510
7.5.10 ssince and suntil . . . 1510
7.5.11 ifinite and inf . . . 1510
7.5.12 future and past len . . . 1511
7.5.13 Validity . . . 1511
7.5.14 Defs . . . 1511
7.6 Time reversal . . . 1516
7.7 Definitions . . . 1538
7.8 Lemmas . . . 1542
7.8.1 nprefix and nprefix_eq . . . 1542
7.8.2 Pref, finitary, finitary_list, infinitary_list, inter_list and union_list . . . 1543
7.8.3 cat and wcat . . . 1549
7.8.4 since . . . 1549
7.8.5 infinitely many and infinite_list_of . . . 1553
7.8.6 Ap, Ep, Rp and Pp . . . 1567
7.8.7 safety and guarantee . . . 1577
7.8.8 response and persistence . . . 1578
7.8.9 obligation . . . 1581
7.8.10 reactivity . . . 1599
7.8.11 Safety and liveness . . . 1613

Chapter 1

Model: Intervals

Intervals are defined in terms of non empty coinductive lists.

1.1 Extra operations on coinductive lists

the operations kfilter, lleast, lbutlast, lidx, ridx, lfirst, lfuse, lsub, lsubc, llastlfirst, lltl, llbutlast, is_lfirst, lfusecat and lrev on coinductive lists are defined together with a library of lemmas.

theory *LList-Extras*

imports
Coinductive.Coinductive-List
begin

definition *kfilter* :: $('a \Rightarrow bool) \Rightarrow nat \Rightarrow 'a\ llist \Rightarrow nat\ llist$
where $kfilter\ P\ n\ xs = lmap\ snd\ (lfilter\ (P \circ fst)\ (lzip\ xs\ (iterates\ Suc\ n)))$

definition *lleast* :: $('a \Rightarrow bool) \Rightarrow 'a\ llist \Rightarrow nat$
where $lleast\ P\ xs = (LEAST\ na.\ na < llength\ xs \wedge P\ (lnth\ xs\ na))$

primcorec *lbutlast* :: $'a\ llist \Rightarrow 'a\ llist$
where *lbutlast xs* =
(*case xs of LNil* ⇒ *LNil* |
(*LCons x xs'*) ⇒
(*case xs' of LNil* ⇒ *LNil* |
(*LCons x1 xs1*) ⇒ (*LCons x* (*lbutlast xs'*))))

simps-of-case *lbutlast-simps* [*simp*, *code*, *nitpick-simp*]: *lbutlast.code*

definition *ridx* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ llist \Rightarrow bool$
where $ridx\ R\ xs = (\forall\ i.\ (Suc\ i) < llength\ xs \longrightarrow R\ (lnth\ xs\ i)\ (lnth\ xs\ (Suc\ i)))$

definition *lidx* :: $nat\ llist \Rightarrow bool$
where $lidx\ xs \longleftrightarrow (\forall n.\ (Suc\ n) < llength\ xs \longrightarrow lnth\ xs\ n < lnth\ xs\ (Suc\ n))$

definition *lfirst* :: *'a llist ⇒ 'a*
where *lfirst xs = lhd xs*

definition *lfuse* :: *'a llist ⇒ 'a llist ⇒ 'a llist*
where *lfuse xs ys =*
(if ¬ lnull xs ∧ ¬ lnull ys then lappend xs (ltl ys) else lappend xs ys)

definition *lsub* :: *nat ⇒ nat ⇒ 'a llist ⇒ 'a llist*
where *lsub k n xs = (ltake (eSuc (n − k)) (ldrop k xs))*

definition *lsubc* :: *nat ⇒ nat ⇒ 'a llist ⇒ 'a llist*
where *lsubc k n xs = (ltake (eSuc (n − k)) (ldrop ((min k (epred (llength xs)))) xs))*

definition *llastlfirst* :: *'a llist llist ⇒ bool*
where *llastlfirst xss = (∀ i. (Suc i)<llength xss ⟶ llast (lnth xss i) = lfirst(lnth xss (Suc i)))*

definition *lltl* :: *'a llist llist ⇒ 'a llist llist*
where *lltl xss = lmap ltl xss*

definition *llbutlast* :: *'a llist llist ⇒ 'a llist llist*
where *llbutlast xss = lmap lbutlast xss*

abbreviation *is-lfirst* ≡ *(λxs. ∃ b. xs = (LCons b LNil))*

definition *lrev* :: *'a llist ⇒ 'a llist*
where *lrev xs = (if lfinite xs then llist-of (rev (list-of xs)) else xs)*

1.1.1 Auxiliary lemmas

lemma *enat-min*:
assumes *m ≥ enat n′*
and *enat n < m − enat n′*
shows *enat n + enat n′ < m*
using *assms*
by *(metis add.commute enat.simps(3) enat-add-mono enat-add-sub-same le-iff-add)*

lemma *enat-min-eq*:
assumes *m ≥ enat n′*
and *enat n ≤ m − enat n′*
shows *enat n + enat n′ ≤ m*
using *assms*
by *(metis add.commute enat.simps(3) enat-add-sub-same enat-min le-iff-add le-less)*

lemma *llist-eq-lnth-eq*:
(xs = ys) ⟷ (llength xs = llength ys ∧ (∀ i < llength xs. lnth xs i = lnth ys i))
proof *auto*
show *llength xs = llength ys ⟹ ∀ i. enat i < llength ys ⟶ lnth xs i = lnth ys i ⟹ xs = ys*
proof *(coinduction arbitrary: xs ys)*

case (*Eq-llist xsa ysa*)
then show *?case*
proof −
assume *a*: *llength xsa* = *llength ysa*
assume *b*: ∀ *i*. *enat i* < *llength ysa* ⟶ *lnth xsa i* = *lnth ysa i*
have *1*: *lnull xsa* = *lnull ysa*
using *a* **by** *auto*
have *2*: ¬ *lnull xsa* ⟶
¬ *lnull ysa* ⟶
lhd xsa = *lhd ysa*
using *b lhd-conv-lnth zero-enat-def* **by** *force*
have *3*: ¬ *lnull xsa* ⟶
¬ *lnull ysa* ⟶
(∃ *xs ys*. *ltl xsa* = *xs* ∧ *ltl ysa* = *ys* ∧ *llength xs* = *llength ys* ∧
(∀ *i*. *enat i* < *llength ys* ⟶ *lnth xs i* = *lnth ys i*))
by (*metis Extended-Nat.eSuc-mono a b eSuc-enat eSuc-epred llength-eq-0 llength-ltl lnth-ltl*)
show *?thesis* **using** *1 2 3* **by** *blast*
qed
qed
qed

lemma *exist-lset-lnth*:
(∃ *x* ∈ *lset xs*. *P x*) ⟷ (∃ *i*<*llength xs*. *P* (*lnth xs i*))
by (*metis in-lset-conv-lnth*)

lemma *exist-llength-gr-zero*:
assumes (∃ *x* ∈ *lset xs*. *P x*)
shows *0*< *llength xs*
by (*metis assms gr-implies-not-zero gr-zeroI in-lset-conv-lnth*)

1.1.2 lbutlast

lemma *lbutlast-snoc* [*simp*]:
lbutlast (*lappend xs* (*LCons x LNil*)) = *xs*
proof (*cases lfinite xs*)
case *True*
then show *?thesis*
proof (*induct rule*: *lfinite-induct*)
case (*LNil xs*)
then show *?case* **using** *llist.collapse*(*1*) **by** *fastforce*
next
case (*LCons xs*)
then show *?case*
proof (*cases xs*=*LNil*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then show *?thesis*
proof −

```
      have 1: ∃ y ys. xs =(LCons y ys)
        using False llist.exhaust-sel by blast
      obtain y ys where 2:  xs =(LCons y ys)
         using 1 by blast
      have 3:  lbutlast (lappend xs (LCons x LNil)) = lbutlast (lappend (LCons y ys) (LCons x LNil))
         by (simp add: 2)
       have 4: lbutlast (lappend (LCons y ys) (LCons x LNil)) = lbutlast (LCons y (lappend ys (LCons x
LNil)))
        by simp
      have 5: lnull ys ⟹ ?thesis
        by (simp add: 2 lnull-def)
      have 6: ¬lnull ys ⟹ ?thesis
         by (metis 2 LCons.hyps(3) eq-LConsD lappend-code(2) lbutlast-simps(3) lhd-LCons-ltl)
      show ?thesis
      using 5 6 by blast
     qed
    qed
  qed
next
case False
then show ?thesis
 proof (coinduction arbitrary: xs)
 case (Eq-llist xsa)
 then show ?case
    proof −
     have 1: lnull (lbutlast (lappend xsa (LCons x LNil))) = lnull xsa
       by (metis Eq-llist lappend-inf lbutlast.disc(2) lfinite.simps lhd-LCons-ltl lnull-imp-lfinite)
     have 2: ¬ lnull (lbutlast (lappend xsa (LCons x LNil))) ⟶
              ¬ lnull xsa ⟶
              lhd (lbutlast (lappend xsa (LCons x LNil))) = lhd xsa
         by (metis Eq-llist eq-LConsD lappend-inf lbutlast.disc(1) lbutlast-simps(3) not-lnull-conv)
     have 3: ¬ lnull (lbutlast (lappend xsa (LCons x LNil))) ⟶
              ¬ lnull xsa ⟶
             (∃xs. ltl (lbutlast (lappend xsa (LCons x LNil))) =
                        lbutlast (lappend xs (LCons x LNil)) ∧ ltl xsa = xs ∧ ¬ lfinite xs)
       by (metis Eq-llist eq-LConsD lappend-inf lbutlast-simps(3) lfinite-ltl lhd-LCons-ltl lnull-imp-lfinite)
     show ?thesis using 1 2 3 by auto
   qed
 qed
qed
```

```
lemma lbutlast-ltl:
  lbutlast (ltl xs) = ltl (lbutlast xs)
proof (cases lfinite xs)
case True
then show ?thesis
  proof  (induct rule: lfinite-induct)
  case (LNil xs)
```

```
  then show ?case by (simp add: lbutlast.ctr(1))
  next
  case (LCons xs)
  then show ?case
    by (simp add: lbutlast.code llist.case-eq-if)
  qed
next
case False
then show ?thesis
   proof  (coinduction arbitrary: xs)
   case (Eq-llist xsa)
   then show ?case
     proof −
      have 1: lnull (lbutlast (ltl xsa)) = lnull (ltl (lbutlast xsa))
        by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
      have 2: ¬ lnull (lbutlast (ltl xsa)) ⟶
                ¬ lnull (ltl (lbutlast xsa)) ⟶
                lhd (lbutlast (ltl xsa)) = lhd (ltl (lbutlast xsa))
        by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
      have 3: ¬ lnull (lbutlast (ltl xsa)) ⟶
                ¬ lnull (ltl (lbutlast xsa)) ⟶
                (∃ xs. ltl (lbutlast (ltl xsa)) = lbutlast (ltl xs) ∧
                      ltl (ltl (lbutlast xsa)) = ltl (lbutlast xs) ∧ ¬ lfinite xs)
        by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
      show ?thesis using 1 2 3 by auto
    qed
   qed
qed

lemma lbutlast-not-lfinite:
 assumes ¬ lfinite xs
 shows   lbutlast xs = xs
 using assms
by (coinduction arbitrary: xs)
   (metis lappend-inf lbutlast-snoc lfinite-ltl)

lemma lbutlast-lfinite:
 lfinite (lbutlast xs) ⟷ lfinite xs
proof
 show lfinite (lbutlast xs) ⟹ lfinite xs
  proof (induct zs≡lbutlast xs rule: lfinite-induct)
  case LNil
  then show ?case using lbutlast-not-lfinite by fastforce
  next
  case LCons
  then show ?case using lbutlast-not-lfinite by fastforce
  qed
 show lfinite xs ⟹ lfinite (lbutlast xs)
  proof (induct rule: lfinite-induct)
  case (LNil xs)
```

```
  then show ?case by simp
  next
  case (LCons xs)
  then show ?case by (simp add: lbutlast-ltl)
  qed
qed

lemma llength-lbutlast [simp]:
  llength (lbutlast xs) = epred (llength xs)
by (coinduction arbitrary: xs rule: enat-coinduct)
   (simp,
   metis epred-enat-unfold  lbutlast-ltl llength-def llength-eq-0 llength-ltl )

lemma lbutlast-lappend:
  lbutlast (lappend xs ys) = (if ys = LNil then lbutlast xs else lappend xs (lbutlast ys))
proof (cases lfinite xs)
case True
then show ?thesis
   proof (induct arbitrary: ys rule: lfinite-induct)
   case (LNil xs)
   then show ?case by (simp add: lnull-def)
   next
   case (LCons xs)
   then show ?case
      proof (cases lnull ys )
      case True
      then show ?thesis by (metis lappend-LNil2 lnull-def)
      next
      case False
      then show ?thesis
        proof (cases lnull xs)
        case True
        then show ?thesis
        using LCons.hyps(2) by auto
        next
        case False
        then show ?thesis using LCons by (auto simp add: llist.case-eq-if not-lnull-conv)
           (metis lappend.ctr(2) lbutlast-simps(3) llist.collapse(1) ltl-simps(2))
        qed
      qed
   qed
next
case False
then show ?thesis by (metis lappend-inf lbutlast-snoc)
qed

lemma lappend-lbutlast-llast-id-lfinite:
 assumes lfinite xs
         ¬ lnull xs
 shows   (lappend (lbutlast xs) (LCons (llast xs) LNil)) = xs
```

using *assms*
proof (*induct rule*: *lfinite-induct*)
case (*LNil xs*)
then show *?case* **by** *simp*
next
case (*LCons xs*)
then show *?case*
  proof (*cases xs*)
  case *lfinite-LNil*
  then show *?thesis* **using** *LCons* **by** *simp*
  next
  case (*lfinite-LConsI xs x*)
  then show *?thesis* **using** *LCons*
  by (*auto simp add*: *llast-LCons*)
    (*metis lappend-code*(*1*) *lbutlast.ctr*(*1*) *llist.collapse*(*1*) *ltl-simps*(*2*),
     metis lappend-code(*2*) *lbutlast-simps*(*3*) *lhd-LCons-ltl*)
qed
qed

lemma *lappend-lbutlast-llast-id-not-lfinite*:
 assumes ¬*lfinite xs*
        ¬ *lnull xs*
 shows (*lappend* (*lbutlast xs*) (*LCons* (*llast xs*) *LNil*)) = *xs*
using *assms*
proof (*coinduction arbitrary*: *xs*)
case (*Eq-llist xsa*)
then show *?case*
  by (*auto simp add*: *lbutlast-ltl llast-linfinite*)
    (*metis lfinite-LNil lfinite-ltl llist.collapse*(*1*),
     metis lbutlast.simps(*3*) *lbutlast-not-lfinite*)
qed

lemma *lappend-lbutlast-llast-id* [*simp*]:
shows ¬ *lnull xs* ⟹ (*lappend* (*lbutlast xs*) (*LCons* (*llast xs*) *LNil*)) = *xs*
using *lappend-lbutlast-llast-id-lfinite lappend-lbutlast-llast-id-not-lfinite* **by** *blast*

lemma *lbutlast-eq-LNil-conv*:
  lbutlast xs = *LNil* ⟷ *xs* =*LNil* ∨ (∃ *x*. *xs* = (*LCons x LNil*))
by (*metis lbutlast.disc-iff*(*1*) *lbutlast-simps*(*2*) *lhd-LCons-ltl llist.collapse*(*1*) *llist.disc*(*1*))

lemma *lbutlast-eq-LCons-conv*:
  lbutlast xs = (*LCons x ys*) ⟷ *xs* = (*LCons x* (*lappend ys* (*LCons* (*llast xs*) *LNil*)))
by (*metis eq-LConsD lappend-code*(*2*) *lappend-lbutlast-llast-id lbutlast.ctr*(*1*) *lbutlast-snoc*)

lemma *lbutlast-conv-ltake*:
  lbutlast xs = *ltake* (*epred* (*llength xs*)) *xs*
proof (*cases lfinite xs*)
case *True*

then show *?thesis*
  proof (*induct rule*: *lfinite-induct*)
  case (*LNil xs*)
  then show *?case* **by** *simp*
  next
  case (*LCons xs*)
  then show *?case*
   by (*metis enat-le-plus-same*(*2*) *gen-llength-def lappend-lbutlast-llast-id-lfinite llength-code*
       llength-lbutlast ltake-all ltake-lappend1)
  qed
next
case *False*
then show *?thesis*
 proof (*coinduction arbitrary*: *xs*)
 case (*Eq-llist xsa*)
 then show *?case*
  by (*auto simp add*: *not-lfinite-llength lbutlast-not-lfinite*)
    (*metis epred-Infty epred-llength lbutlast-not-lfinite llength-eq-infty-conv-lfinite ltake-epred-ltl*)
 qed
qed

lemma *lmap-lbutlast*:
 lmap f (*lbutlast xs*) = *lbutlast* (*lmap f xs*)
by (*simp add*: *lbutlast-conv-ltake*)

lemma *snocs-eq-iff-lbutlast*:
 lappend xs (*LCons x LNil*) = *ys* ⟷
  ((*lfinite ys* ∧ ¬ *lnull ys* ∧ *lbutlast ys* = *xs* ∧ *llast ys* = *x*)
  ∨ (¬ *lfinite ys* ∧ *lbutlast xs* = *ys*))
by (*metis lappend.disc*(*2*) *lappend-inf lappend-lbutlast-llast-id lbutlast.ctr*(*1*) *lbutlast-snoc*
    lfinite-LNil llast-lappend llast-singleton llist.discI(*2*))

lemma *in-lset-lbutlastD*:
 x ∈ *lset*(*lbutlast xs*) ⟹ *x* ∈ *lset xs*
by (*metis in-lset-lappend-iff lappend-ltake-ldrop lbutlast-conv-ltake*)

lemma *in-lset-lbutlast-lappendI*:
 x ∈ *lset* (*lbutlast xs*) ∨ (*lfinite xs* ∧ *x* ∈ *lset*(*lbutlast ys*)) ⟹
  x ∈ *lset* (*lbutlast* (*lappend xs ys*))
by (*metis empty-iff in-lset-lappend-iff in-lset-lbutlastD lbutlast-lappend lset-LNil*)

lemma *lnth-lbutlast*:
assumes *n*< *llength*(*lbutlast xs*)
shows *lnth* (*lbutlast xs*) *n* = *lnth xs n*
proof (*cases xs*)
case *LNil*
then show *?thesis* **by** *simp*
next
case (*LCons x21 x22*)
then show *?thesis* **by** (*metis assms lbutlast-conv-ltake llength-lbutlast lnth-ltake*)

qed

1.1.3 lfuse

lemma *lfuse-conv-lnull*:
lnull (*lfuse xs ys*) ⟷ *lnull xs* ∧ *lnull ys*
by (*simp add*: *lfuse-def*)

lemma *lfuse-LNil-1* [*simp*]:
lfuse LNil ys = *ys*
by (*simp add*: *lfuse-def*)

lemma *lfuse-LNil-2* [*simp*]:
lfuse xs LNil = *xs*
by (*metis lappend-lnull2 lfuse-def llist.discI*(*1*))

lemma *lfuse-One-a* [*simp*]:
assumes ¬ *lnull xs*
shows *lfuse* (*LCons* (*lhd xs*) *LNil*) *xs* = *xs*
using *assms* **by** (*simp add*: *lfuse-def*)

lemma *lfuse-One-b* [*simp*]:
assumes ¬ *lnull xs*
shows *lfuse xs* (*LCons y LNil*) = *xs*
using *assms*
by (*simp add*: *lfuse-def*)

lemma *lfuse-One-a-var*:
shows *lfuse* (*LCons x LNil*) *ys* = (*if* ¬ *lnull ys then* (*LCons x* (*ltl ys*)) *else* (*LCons x LNil*))
unfolding *lfuse-def* **by** *simp*

lemma *lfuse-One-b-var*:
shows *lfuse xs* (*LCons y LNil*) = (*if* ¬*lnull xs then xs else* (*LCons y LNil*))
unfolding *lfuse-def* **by** (*simp add*: *lappend-lnull1*)

lemma *lfuse-LCons-a* [*simp*]:
lfuse (*LCons x xs*) *ys* =
(*if lnull xs then* (*LCons x* (*ltl ys*)) *else* (*LCons x* (*lfuse xs ys*)))
by (*metis lappend-code*(*2*) *lappend-lnull1 lfuse-def llist.collapse*(*1*) *llist.disc*(*2*) *ltl-simps*(*1*))

lemma *lfuse-LCons-b* [*simp*]:
lfuse xs (*LCons y ys*) =
(*if* ¬ *lnull xs then lappend xs ys else* (*LCons y ys*))
unfolding *lfuse-def* **by** (*simp add*: *lappend-lnull1*)

lemma *lfuse-simps* [*simp*]:
shows *lhd-lfuse*: *lhd* (*lfuse xs ys*) = (*if lnull xs then lhd ys else lhd xs*)
and *ltl-lfuse*: *ltl* (*lfuse xs ys*) =
(*if lnull xs*

then ltl ys
else (if lnull (ltl ys)
then ltl xs
else lappend (ltl xs) (ltl ys)))
by (*auto simp add*: *lfuse-def lappend-lnull2*)

lemma *lfuse-lbutlast*:
assumes *llast xs = lfirst ys*
shows *lfuse xs ys = (if lnull ys then xs else lappend (lbutlast xs) ys)*
using *assms*
by (*metis lappend-lbutlast-llast-id lappend-snocL1-conv-LCons2 lbutlast.ctr(1) lfirst-def*
lfuse-LNil-2 lfuse-def lhd-LCons-ltl llist.collapse(1))

lemma *lfuse-llength*:
shows *llength (lfuse xs ys) = (llength xs) + (if lnull xs then llength ys else epred(llength ys))*
unfolding *lfuse-def* **by** (*simp add*: *llist.case-eq-if epred-llength*)

lemma *not-lnull-llength*:
¬ *lnull xs* ⟷ *1 ≤ llength xs*
by (*metis gr-zeroI ileI1 llength-eq-0 not-one-le-zero one-eSuc*)

lemma *lfuse-llength-atmost-one*:
shows *llength (lfuse xs ys) ≤ 1* ⟷ *llength xs ≤1 ∧ llength ys ≤ 1*
proof (*cases lnull xs*)
case *True*
then show *?thesis*
by (*metis lfuse-LNil-1 llength-LNil lnull-def zero-le*)
next
case *False*
then show *?thesis* **unfolding** *lfuse-llength*
by *auto*
(*meson dual-order.trans enat-le-plus-same(1),*
metis add.commute add-increasing2 co.enat.exhaust-sel not-lnull-llength order-antisym-conv
order-refl plus-1-eSuc(2) zero-le,
metis add-decreasing2 epred-1 epred-le-epredI)
qed

lemma *lfuse-llength-less-a*:
assumes *1 < llength xs*
1 < llength ys
llast xs = lfirst ys
lfinite xs
shows *llength xs < llength (lfuse xs ys)*
unfolding *lfuse-def*
using *assms*
by (*auto simp add*: *lfinite-llength-enat*)
(*metis co.enat.exhaust-sel epred-llength linorder-neq-iff llength-eq-0 one-eSuc*)

lemma *lfuse-llength-less-b*:
assumes *1 < llength xs*

```
        1 < llength ys
        llast xs = lfirst ys
        lfinite ys
 shows    llength ys < llength (lfuse xs ys)
proof −
 have 1: lfuse xs ys = lappend (lbutlast xs) ys
  by (metis assms(2) assms(3) lfuse-lbutlast llength-lnull not-less-zero)
 have 2:  llength (lappend (lbutlast xs) ys) = epred(llength xs) + llength ys
   by simp
 have 3: 0 < epred(llength xs)
   by (metis assms(1) co.enat.exhaust-sel gr-zeroI less-numeral-extra(4) not-one-less-zero one-eSuc)
 have 4: llength ys < epred(llength xs) + llength ys
   using 3 assms(4) lfinite-llength-enat by auto
 show ?thesis
 using 1 2 4 by presburger
qed

lemma lfuse-llength-le-a:
 llength xs ≤ llength (lfuse xs ys)
by (simp add: lfuse-llength)

lemma lfuse-llength-le-b:
 assumes llast xs = lfirst ys
 shows   llength ys ≤ llength (lfuse xs ys)
by (simp add: assms lfuse-lbutlast)

lemma lfuse-lnth-a:
 assumes (enat i) < llength xs
 shows   lnth (lfuse xs ys) i = lnth xs i
using assms
by (simp add: lfuse-def lnth-lappend1)

lemma lfuse-lnth-b:
 assumes  llength xs ≤ (enat i)
          (enat i) < llength (lfuse xs ys)
 shows   lnth (lfuse xs ys) i = (lnth ys (i − (the-enat(epred(llength xs)))))
proof (cases ¬ lnull xs ∧ ¬lnull ys)
case True
then show ?thesis
   proof −
    have 1: lfinite xs
      using assms(1) lfinite-ldropn lnull-imp-lfinite lnull-ldropn by blast
    obtain n where 15: (enat n) = llength xs using 1 by (metis assms(1) enat-ile)
    have 16: (the-enat(epred(llength xs))) = n−1
      by (metis 15 epred-enat the-enat.simps)
    have 17: 0<n
      by (metis 15 True gr0I llength-eq-0 zero-enat-def)
    have 2: lfuse xs ys = lappend xs (ltl ys)
      unfolding lfuse-def using assms True by presburger
    have 3: n−1 ≤ i
```

```
      by (metis 15 17 Suc-pred' assms(1) enat-ord-simps(2) leD le-imp-less-Suc wlog-linorder-le)
    have 4: (Suc (i − n)) =  (i − (n−1))
      by (metis 15 17 Suc-diff-eq-diff-pred Suc-diff-le assms(1) enat-ord-simps(1))
    have 5: lnth (ltl ys) (i − the-enat (llength xs)) = (lnth ys (i − (the-enat(epred(llength xs)))))
      using True 3 lnth-ltl[of ys (i − the-enat (llength xs))] by (metis 15 16 4 the-enat.simps)
   show ?thesis   using lnth-lappend[of xs (ltl ys) ] by (simp add: 2 5 assms(1) leD)
  qed
next
case False
then show ?thesis  using assms unfolding lfuse-def
using lappend-lnull1 by fastforce
qed

lemma lfuse-lnth-c:
 assumes  epred(llength xs) ≤ (enat i)
          (enat i) < llength (lfuse xs ys)
          llast xs = lfirst ys
 shows    lnth (lfuse xs ys) i =
          (if lnull ys then lnth xs (the-enat (epred(llength xs)))
           else   lnth ys (i − (the-enat (epred(llength xs)))))
proof (cases ¬ lnull xs ∧ ¬lnull ys)
case True
then show ?thesis
  using assms
  by (simp add: leD lfuse-lbutlast lnth-lappend)
next
case False
then show ?thesis
   proof (cases lnull xs)
   case True
   then show ?thesis
      using assms
     by (metis epred-0 gr-implies-not-zero lfuse-conv-lnull lfuse-lnth-b llength-lnull)
   next
   case False
   then show ?thesis
      using assms
      by (metis co.enat.exhaust-sel iless-Suc-eq leD lfuse-lbutlast llength-eq-0
          llength-lbutlast lnth-lappend nle-le the-enat.simps)
   qed
qed


lemma lfirst-lfuse-1:
 shows    lfirst (lfuse xs ys) = (if ¬ lnull xs then lfirst xs else lfirst ys)
by (simp add: lfirst-def)

lemma llast-lfuse:
 assumes  ¬ lnull xs
          ¬ lnull ys
```

lfinite xs
lfinite ys
llast xs = lfirst ys
shows *llast(lfuse xs ys) = llast ys*
using *assms*
by (*metis lfirst-def lfuse-def lhd-LCons-ltl llast-LCons llast-lappend*)

lemma *lfuse-assoc*:
assumes ¬ *lnull xs*
¬ *lnull ys*
¬ *lnull zs*
shows (*lfuse xs* (*lfuse ys zs*)) = (*lfuse* (*lfuse xs ys*) *zs*)
using *assms*
by (*metis lappend-assoc lappend-ltl lfuse-def lfuse-conv-lnull*)

lemma *lfirst-llast*:
assumes *i< llength xs*
shows *llast* (*ltake* (*Suc i*) *xs*) = *lfirst* (*ldropn i xs*)
using *assms*
by (*simp add*: *lfirst-def lhd-ldropn ltake-Suc-conv-snoc-lnth*)

lemma *ltake-lfuse*:
shows *ltake* (*llength xs*) (*lfuse xs ys*) = *xs*
by (*metis dual-order.refl lappend-lnull2 lfuse-def ltake-all ltake-lappend1*)

lemma *llast-lfirst-LNil*:
llast LNil = lfirst LNil
by (*simp add*: *lfirst-def lhd-def llast-LNil*)

lemma *ldrop-lappend-either-LNil*:
assumes *lnull xs* ∨ *lnull ys*
shows *ldrop* (*llength xs*) (*lappend xs ys*) =
(*if lfinite xs then ys else LNil*)
proof −
have *1*: *lnull xs* ⟹ *?thesis*
by (*simp add*: *lappend-lnull1*)
have *2*: *lnull ys* ⟹ *?thesis*
by (*metis dual-order.refl lappend-LNil2 ldrop-eq-LNil lnull-def*)
show *?thesis*
using *1 2 assms* **by** *blast*
qed

lemma *ldrop-lfuse*:
assumes *llast xs = lfirst ys*
shows *ldrop* (*if* ¬ *lnull xs* ∧ ¬ *lnull ys then epred*(*llength xs*) *else* (*llength xs*)) (*lfuse xs ys*) =
(*if lfinite xs then ys else LNil*)
proof −
have *3*: *lfuse xs ys* = (*if* ¬ *lnull xs* ∧ ¬ *lnull ys*
then lappend (*lbutlast xs*) *ys*
else lappend xs ys)

by (*meson assms lfuse-def lfuse-lbutlast*)
have *4*: *ldrop* ((*if* ¬ *lnull xs* ∧ ¬ *lnull ys then epred*(*llength xs*) *else* (*llength xs*)))
(*if* ¬ *lnull xs* ∧ ¬ *lnull ys*
then lappend (*lbutlast xs*) *ys*
else lappend xs ys) = (*if lfinite xs then ys else LNil*)
proof (*cases lfinite xs*)
case *True*
then show *?thesis*
by (*simp add*: *ldrop-lappend lfinite-llength-enat*)
next
case *False*
then show *?thesis*
by *simp*
(*metis epred-Infty llength-eq-infty-conv-lfinite lnull-imp-lfinite*)
qed
show *?thesis*
by (*simp add*: *3 4*)
qed

lemma *ldrop-lfuse-a*:
assumes ¬ *lnull xs*
¬ *lnull ys*
llast xs = *lfirst ys*
shows *ldrop* (*epred*(*llength xs*)) (*lfuse xs ys*) =
(*if lfinite xs then ys else LNil*)
using *assms*
using *ldrop-lfuse* **by** *force*

lemma *lfuse-ltake-ldrop*:
assumes *i* < *llength xs*
shows *lfuse* (*ltake* (*eSuc i*) *xs*) (*ldrop i xs*) = *xs*
using *assms*
by (*metis lappend-ltake-ldrop ldrop-eSuc-conv-ltl ldrop-lnull leD lfuse-def lnull-ldrop*
ltake.disc(*2*) *zero-ne-eSuc*)

lemma *lset-lfuse*:
shows *lset* (*lfuse xs ys*) =
(*if lfinite xs then*
(*if* ¬ *lnull xs* ∧ ¬ *lnull ys then lset xs* ∪ *lset* (*ltl ys*) *else lset xs* ∪ *lset ys*)
else lset xs)
by (*simp add*: *lappend-inf lfuse-def*)

lemma *mono-llist-lfuse2* [*partial-function-mono*]:
mono-llist A ⟹ *mono-llist* (λ*f*. *lfuse xs* (*A f*))
by (*auto intro*!: *monotoneI lprefix-lappend-sameI simp add*: *lfuse-def lnull-lprefix*

llist.case-eq-if fun-ord-def lprefix-ltlI monotone-def dest: *monotoneD*)
(*metis llist.collapse*(*1*) *lprefix-ltlI ltl-simps*(*1*))

lemma *mono2mono-lfuse2* [*THEN llist.mono2mono*, *cont-intro*, *simp*]:
shows *monotone-lfuse2*: *monotone* (*lprefix*) (*lprefix*) (*lfuse xs*)
by (*rule monotoneI*)
(*metis Coinductive-List.finite-lprefix-def Coinductive-List.finite-lprefix-nitpick-simps*(*1*)
lfuse-def lprefix-LNil lprefix-lappend-sameI ltl-simps(*1*) *monotoneD monotone-ltl*)

lemma *silys*:
assumes $f = g$
shows *mcont lSup* (*lprefix*) *lSup* (*lprefix*) f =
mcont lSup (*lprefix*) *lSup* (*lprefix*) g
using *assms* **by** *auto*

lemma *lfuse-LNil-eq-id*:
lfuse LNil = *id*
by (*simp add*: *fun-eq-iff llist.case-eq-if*)

lemma *mcont2mcont-lfuse2* [*THEN llist.mcont2mcont*, *cont-intro*, *simp*]:
shows *mcont-lfuse2*: *mcont lSup* (*lprefix*) *lSup* (*lprefix*) (*lfuse xs*)
proof(*cases lfinite xs*)
case *True*
thus *?thesis*
proof *induct*
case *lfinite-LNil*
then show *?case* **using** *lfuse-LNil-eq-id* **by** *simp*
next
case (*lfinite-LConsI xs x*)
then show *?case* **by** (*simp add*:*monotone-lfuse2*)
qed
next
case *False*
hence *lfuse xs* = (λ-. *xs*)
by (*simp add*: *fun-eq-iff lappend-inf lfuse-def*)
thus *?thesis* **by**(*simp add*: *ccpo.cont-const*[*OF llist-ccpo*])
qed

lemma *lfuse-inf*: ¬ *lfinite xs* ⟹ *lfuse xs ys* = *xs*
by (*simp add*: *lappend-inf lfuse-def*)

lemma *llist-all2-lfuseI*:
assumes *1*: *llist-all2 P xs ys*
and *2*: ⟦ *lfinite xs*; *lfinite ys* ⟧ ⟹ *llist-all2 P xs′ ys′*
shows *llist-all2 P* (*lfuse xs xs′*) (*lfuse ys ys′*)
proof(*cases lfinite xs*)

case *True*
with *1* **have** *lfinite ys* **by**(*auto dest*: *llist-all2-lfiniteD*)
from *1 2*[*OF True this*] **show** *?thesis*
proof (*coinduction arbitrary*: *xs ys*)
case *LNil*
then show *?case* **by** (*simp add*: *lfuse-conv-lnull llist-all2-lnullD*)
next
case *LCons*
then show *?case*
by (*metis* (*no-types*, *lifting*) *lfuse-def llist.rel-sel llist-all2-lappendI*)
qed
next
case *False*
with *1* **have** ¬ *lfinite ys* **by**(*auto dest*: *llist-all2-lfiniteD*)
with *False 1* **show** *?thesis*
by (*simp add*: *lfuse-inf*)
qed

1.1.4 ridx and lidx

lemma *ridx-lidx*:
ridx (<) *xs* = *lidx xs*
unfolding *lidx-def ridx-def*
by *simp*

lemma *ridx-expand-1*:
ridx R xs ⟷ *lnull xs* ∨ *llength xs* = *1* ∨
(*1* < *llength xs* ∧ (∀ *n*. (*Suc n*)<*llength xs* ⟶ *R* (*lnth xs n*) (*lnth xs* (*Suc n*))))
unfolding *ridx-def*
by (*metis Zero-not-Suc enat-0-iff*(*1*) *gr-implies-not-zero iless-eSuc0 linorder-cases llength-eq-0 one-eSuc*)

lemma *lidx-expand-1*:
lidx xs ⟷ *lnull xs* ∨ *llength xs* = *1* ∨
(*1* < *llength xs* ∧ (∀ *n*. (*Suc n*)<*llength xs* ⟶ *lnth xs n* < *lnth xs* (*Suc n*)))
using *ridx-lidx ridx-expand-1* **by** *blast*

lemma *ridx-LCons*:
ridx R (*LCons x xs*) ⟷
lnull xs ∨
(*0* < *llength xs* ∧
(∀ *n*. (*Suc n*)< *eSuc* (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))))
unfolding *ridx-def*
using *enat-0-iff*(*1*) **by** *force*

lemma *lidx-LCons*:
lidx (*LCons x xs*) ⟷
lnull xs ∨
(*0* < *llength xs* ∧
(∀ *n*. (*Suc n*)< *eSuc* (*llength xs*) ⟶ *lnth* (*LCons x xs*) *n* < *lnth* (*LCons x xs*) (*Suc n*)))

using *ridx-lidx*[*of* (*LCons x xs*)] *ridx-LCons*[*of* (<) *x xs*] **by** *blast*

lemma *ridx-LCons-conv*:
(*0* < *llength xs* ∧
(∀ *n*. (*Suc n*)< *eSuc* (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))))
⟷ (*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *0*≤ *n* ∧ (*Suc n*)< (*llength xs*) ⟶ *R* (*lnth xs n*) (*lnth xs* (*Suc n*))))
proof −
have *1*: (*0* < *llength xs* ∧
(∀ *n*. (*Suc n*)< *eSuc* (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))) ⟷
(*0* < *llength xs* ∧
(∀ *n*. *0*≤ *n* ∧ (*Suc n*)< *eSuc* (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))))
by *blast*
have *2*: (*0* < *llength xs* ∧
(∀ *n*. *0*≤ *n* ∧ (*Suc n*)< *eSuc* (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))) ⟷
(*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *1*≤ *n* ∧ (*Suc n*)< *eSuc* (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))))
by (*metis Extended-Nat.eSuc-mono One-nat-def eSuc-enat gr-implies-not-zero ldropn-0 less-Suc-eq*
lhd-ldropn lnth-0 lnth-Suc-LCons not-le-imp-less zero-enat-def)
have *3*: (*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *1*≤ *n* ∧ (*Suc n*)< *eSuc* (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))) ⟷
(*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *1*≤ *n* ∧ (*n*)< (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))))
using *Suc-ile-eq* **by** *auto*
have *4*: (*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *1*≤ *n* ∧ (*n*)< (*llength xs*) ⟶ *R* (*lnth* (*LCons x xs*) *n*) (*lnth* (*LCons x xs*) (*Suc n*))) ⟷
(*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *1*≤ *n* ∧ (*n*)< (*llength xs*) ⟶ *R* (*lnth xs* (*n* −*1*)) (*lnth xs* (*n*))))
by (*metis le-add-diff-inverse llist.disc*(*2*) *lnth-ltl ltl-simps*(*2*) *plus-1-eq-Suc*)
have *5*: (*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *1*≤ *n* ∧ (*n*)< (*llength xs*) ⟶ *R* (*lnth xs* (*n* −*1*)) (*lnth xs* (*n*))) ⟷
(*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *0*≤ (*n*−*1*) ∧ (*Suc* (*n*−*1*))< (*llength xs*) ⟶ *R* (*lnth xs* (*n* −*1*)) (*lnth xs* (*Suc* (*n* −*1*)))))
by (*metis diff-Suc-1 le0 le-add1 le-add-diff-inverse plus-1-eq-Suc*)
have *6*: (*0* < *llength xs* ∧
R x (*lhd xs*) ∧
(∀ *n*. *0*≤ (*n*−*1*) ∧ (*Suc* (*n*−*1*))< (*llength xs*) ⟶ *R* (*lnth xs* (*n* −*1*)) (*lnth xs* (*Suc* (*n* −*1*)))))

$\longleftrightarrow$
(*0 < llength xs* $\wedge$
R x (*lhd xs*) $\wedge$
($\forall$ *n. 0*$\leq$ *n* $\wedge$ (*Suc n*)$<$ (*llength xs*) $\longrightarrow$ *R* (*lnth xs n*) (*lnth xs* (*Suc n*))))
by (*metis diff-Suc-1*)
show *?thesis*
using *2 3 4 5 6* **by** *auto*
qed

lemma *lidx-LCons-conv*:
(*0 < llength xs* $\wedge$ ($\forall$ *n.* (*Suc n*)$<$ *eSuc* (*llength xs*) $\longrightarrow$ *lnth* (*LCons x xs*) *n* $<$ *lnth* (*LCons x xs*) (*Suc n*)))
$\longleftrightarrow$ (*0 < llength xs* $\wedge$
x$<$ *lhd xs* $\wedge$
($\forall$ *n. 0*$\leq$ *n* $\wedge$ (*Suc n*)$<$ (*llength xs*) $\longrightarrow$ *lnth xs n* $<$ *lnth xs* (*Suc n*)))
using *ridx-LCons-conv*
by *metis*

lemma *ridx-LCons-1* [*simp*]:
ridx R (*LCons x xs*) $\longleftrightarrow$ *lnull xs* $\vee$ (*0 < llength xs* $\wedge$ *R x* (*lhd xs*) $\wedge$ *ridx R xs*)
unfolding *ridx-def*
using *ridx-LCons-conv*[*of xs R x*]
by (*auto simp add*: *enat-0-iff*(*1*))

lemma *lidx-LCons-1* [*simp*]:
lidx (*LCons x xs*) $\longleftrightarrow$ *lnull xs* $\vee$ (*0 < llength xs* $\wedge$ *x*$<$ *lhd xs* $\wedge$ *lidx xs*)
using *ridx-lidx*[*of LCons x xs*] *ridx-lidx*[*of xs*]*ridx-LCons-1*[*of* (<) *x xs*] **by** *blast*

lemma *ridx-less*:
assumes *ridx R xs*
Suc(*n*+*k*) $<$ *llength xs*
transp R
shows *R* (*lnth xs n*) (*lnth xs* (*Suc*(*n*+*k*)))
using *assms* **unfolding** *ridx-def*
proof (*induct k*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
by (*metis Suc-ile-eq add-Suc-right order-less-imp-le transpE*)
qed

lemma *lidx-less*:
assumes *lidx xs*
Suc(*n*+*k*) $<$ *llength xs*
shows *lnth xs n* $<$ *lnth xs* (*Suc*(*n*+*k*))
using *assms ridx-lidx ridx-less*[*of* (<) *xs n k*] *transp-on-less* **by** *blast*

lemma *ridx-less-eq*:
assumes *ridx R xs*
k ≤ *j*
j < *llength xs*
transp R
reflp R
shows *R* (*lnth xs k*) (*lnth xs j*)
proof (*cases k=j*)
case *True*
then show *?thesis* **using** *assms* **by** (*simp add*: *reflp-def*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis less-iff-Suc-add order-neq-le-trans ridx-less*)
qed

lemma *lidx-less-eq*:
assumes *lidx xs*
k ≤ *j*
j < *llength xs*
shows *lnth xs k* ≤ *lnth xs j*
using *assms ridx-lidx ridx-less-eq*[*of* (<) *xs k j*]
by (*metis dual-order.order-iff-strict less-iff-Suc-add lidx-less*)

lemma *ridx-gr-first*:
assumes *ridx R xs*
0<*i*
i < *llength xs*
transp R
shows *R* (*lnth xs 0*) (*lnth xs i*)
using *assms ridx-less*[*of R xs 0 i*−*1*] **unfolding** *ridx-def* **by** *simp*

lemma *lidx-gr-first*:
assumes *lidx xs*
0<*i*
i < *llength xs*
shows (*lnth xs 0*) < *lnth xs i*
using *assms lidx-less*[*of xs 0 i*−*1*] **unfolding** *lidx-def* **by** *simp*

lemma *ridx-ltake-a*:
assumes *ridx R xs*
n ≤ *llength xs*
shows *ridx R* (*ltake n xs*)
using *assms*
unfolding *ridx-def*
by *simp*
(*metis Suc-ile-eq dual-order.strict-trans1 lnth-ltake order-less-imp-le*)

lemma *lidx-ltake-a*:
assumes *lidx xs*

$n \leq$ *llength xs*
shows *lidx* (*ltake n xs*)
using *assms*
using *ridx-lidx ridx-ltake-a* **by** *blast*

lemma *ridx-lappend-lfinite*:
assumes *lfinite xs*
shows *ridx R* (*lappend xs ys*) $\longleftrightarrow$
ridx R xs $\wedge$ ((*lnull xs* $\vee$ *lnull ys*) $\vee$ *R* (*llast xs*) (*lhd ys*)) $\wedge$ *ridx R ys*
using *assms*
proof (*induction xs arbitrary*: *ys*)
case *lfinite-LNil*
then show *?case* **by** (*simp add*: *ridx-expand-1*)
next
case (*lfinite-LConsI xs x*)
then show *?case*
proof (*cases lnull xs*)
case *True*
then show *?thesis*
by (*metis lappend-code*(*1*) *lappend-code*(*2*) *llast-singleton llength-LCons llist.collapse*(*1*)
one-eSuc order-neq-le-trans ridx-LCons-1 ridx-expand-1 zero-le)
next
case *False*
then show *?thesis*
by (*simp add*: *lfinite-LConsI.IH llast-LCons*)
qed
qed

lemma *lidx-lappend-lfinite*:
assumes *lfinite xs*
shows *lidx* (*lappend xs ys*) $\longleftrightarrow$
lidx xs $\wedge$ ((*lnull xs* $\vee$ *lnull ys*) $\vee$ (*llast xs*) $<$ (*lhd ys*)) $\wedge$ *lidx ys*
using *assms* **by** (*metis ridx-lappend-lfinite ridx-lidx*)

lemma *ridx-ldrop*:
assumes *ridx R xs*
$n \leq$ *llength xs*
shows *ridx R* (*ldrop n xs*)
proof −
have *1*: *xs* = *lappend* (*ltake n xs*) (*ldrop n xs*)
by (*simp add*: *lappend-ltake-ldrop*)
have *2*: ¬ *lfinite* (*ltake n xs*) $\Longrightarrow$ *?thesis*
by (*simp add*: *ridx-expand-1*)
have *3*: *lfinite* (*ltake n xs*) $\Longrightarrow$ *ridx R* (*lappend* (*ltake n xs*) (*ldrop n xs*)) $\longleftrightarrow$
ridx R (*ltake n xs*) $\wedge$
((*lnull* (*ltake n xs*) $\vee$ *lnull* (*ldrop n xs*)) $\vee$ *R* (*llast* (*ltake n xs*)) (*lhd* (*ldrop n xs*))) $\wedge$
ridx R (*ldrop n xs*)
using *ridx-lappend-lfinite*[*of* (*ltake n xs*) *R* (*ldrop n xs*)] **by** *blast*
have *4*: *lfinite* (*ltake n xs*) $\Longrightarrow$ *?thesis*
using *1 3 assms* **by** *metis*

show *?thesis* **using** *2 4* **by** *blast*
qed

lemma *lidx-ldrop*:
assumes *lidx xs*
$n \leq$ *llength xs*
shows *lidx* (*ldrop n xs*)
using *assms ridx-ldrop ridx-lidx* **by** *blast*

lemma *ridx-ltake-all*:
assumes $\bigwedge n.$ $n \leq$ *llength xs* $\Longrightarrow$ *ridx R* (*ltake* (*enat n*) *xs*)
shows *ridx R xs*
using *assms*
unfolding *ridx-def*
by *auto*
(*metis Suc-ile-eq dual-order.refl eSuc-enat ile-eSuc lessI lnth-ltake*)

lemma *lidx-ltake-all*:
assumes $\bigwedge n.$ $n \leq$ *llength xs* $\Longrightarrow$ *lidx* (*ltake* (*enat n*) *xs*)
shows *lidx xs*
using *assms ridx-ltake-all ridx-lidx* **by** *blast*

lemma *ridx-ltake*:
assumes *ridx R* (*ltake n xs*)
$n \leq$ *llength xs*
$k \leq n$
shows *ridx R* (*ltake* (*enat k*) *xs*)
using *assms*
using *ridx-ltake-a* **by** *fastforce*

lemma *lidx-ltake*:
assumes *lidx* (*ltake n xs*)
$n \leq$ *llength xs*
$k \leq n$
shows *lidx* (*ltake* (*enat k*) *xs*)
using *assms ridx-ltake ridx-lidx* **by** *blast*

lemma *lidx-imp-lsorted*:
assumes *lidx xs*
shows *lsorted xs*
using *assms*
by (*metis* (*no-types*, *lifting*) *less-imp-le lhd-LCons-ltl lidx-LCons-1 lsorted-coinduct'*)

lemma *lidx-imp-ldistinct*:
assumes *lidx xs*
shows *ldistinct xs*
using *assms*
proof (*coinduction arbitrary*: *xs*)
case (*ldistinct xsa*)
then show *?case*

proof −
have *1*: *lhd xsa* ∉ *lset* (*ltl xsa*)
by (*metis empty-iff lappend-code*(*1*) *lappend-lnull2 ldistinct*(*1*) *ldistinct*(*2*) *leD lhd-LCons-ltl lidx-LCons-1 lidx-imp-lsorted lmember-code*(*2*) *lset-LNil lset-lmember lsortedD*)
have *2*: ((∃ *xs*. *ltl xsa* = *xs* ∧ *lidx xs*) ∨ *ldistinct* (*ltl xsa*))
unfolding *lidx-def*
by (*metis Extended-Nat.eSuc-mono eSuc-enat ldistinct*(*1*) *ldistinct*(*2*) *lhd-LCons-ltl lidx-def llength-LCons lnth-ltl*)
show *?thesis*
using *1 2* **by** *auto*
qed
qed

lemma *ldistinct-Ex1*:
assumes *ldistinct xs*
x ∈ *lset xs*
shows ∃!*i*. *i*< *llength xs* ∧ (*lnth xs i*) = *x*
using *assms*
by (*metis in-lset-conv-lnth ldistinct-conv-lnth*)

lemma *lidx-lset-eq*:
assumes *lidx xs*
lidx ys
lset xs = *lset ys*
shows *xs* = *ys*
using *assms*
by (*simp add*: *lidx-imp-ldistinct lidx-imp-lsorted lsorted-ldistinct-lset-unique*)

lemma *ridx-lfuse-lfirst-llast*:
assumes *ridx R ys*
(*lnth ys 0*) = (*0*::*nat*)
ridx R zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
llast ys = *cp*
shows *llast ys* = *lfirst*(*lmap* (λ*x*. *x*+*cp*) *zs*)
using *assms* **unfolding** *lfirst-def*
by (*simp add*: *lnth-0-conv-lhd*)

lemma *lidx-lfuse-lfirst-llast*:
assumes *lidx ys*
(*lnth ys 0*) = *0*
lidx zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite xs
¬ *lnull xs*

$\neg$ *lnull zs*
llast ys = cp
shows *llast ys = lfirst(lmap* ($\lambda x.$ *x+cp) zs)*
using *assms*
by (*simp add*: *lfirst-def lnth-0-conv-lhd*)

lemma *ridx-lfuse-lnth-cp*:
assumes *ridx R ys*
(lnth ys 0) = 0
ridx R zs
(lnth zs 0) = 0
lfinite ys
lfinite zs
lfinite xs
$\neg$ *lnull xs*
$\neg$ *lnull zs*
$\neg$ *lnull ys*
llast ys = cp
llast zs = the-enat(epred (llength xs)) − cp
i < *(llength zs)*
cp < *llength xs*
shows *lnth (lfuse ys (lmap* ($\lambda x.$ *x+cp) zs)) (the-enat(epred(llength ys)) + i) = cp + (lnth zs i)*
proof −
have *1*: *llast ys = lfirst(lmap* ($\lambda x.$ *x+cp) zs)*
by (*metis assms(11) assms(4) assms(9) lfirst-def llist.map-sel(1) lnth-0-conv-lhd plus-nat.add-0*)
have *3*: *lfirst(lmap* ($\lambda x.$ *x+cp) zs) = cp + (lnth zs 0)*
using *1 assms(11) assms(4)* **by** *force*
have *8*: *i*$\leq$ *epred(llength zs)*
using *assms*
by (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0*)
have *81*: *enat (the-enat (epred (llength ys)) + i)* $\leq$
enat (the-enat (epred (llength ys)) + (the-enat (epred(llength zs))))
by (*metis 8 add-mono-thms-linordered-semiring(2) assms(6) assms(9) co.enat.exhaust-sel enat-ord-simps(1) enat-ord-simps(4) enat-the-enat ile-eSuc iless-Suc-eq lfinite-llength-enat llength-eq-0 order-le-less-trans*)
have *9*: *epred(llength zs)* < *llength zs*
by (*metis assms(6) assms(9) co.enat.exhaust-sel ile-eSuc iless-Suc-eq lfinite-llength-enat llength-eq-0 order-neq-le-trans*)
have *10*: *epred (llength ys)* $\leq$ *enat (the-enat (epred (llength ys)) + (i::nat))*
by (*metis assms(10) assms(5) co.enat.exhaust-sel enat-ord-simps(1) enat-the-enat ile-eSuc infinity-ileE le-add1 lfinite-llength-enat llength-eq-0*)
have *83*: *enat (the-enat (epred (llength ys)) + (the-enat (epred(llength zs)))) =*
enat (the-enat (epred (llength ys) + epred(llength zs)))
by (*metis 10 9 enat-ord-code(4) enat-the-enat leD order-less-imp-le plus-enat-simps(1)*)
have *84*: *enat (the-enat (epred (llength ys) + epred(llength zs))) =*
((epred (llength ys) + epred(llength zs)))
by (*metis 10 9 enat-ord-code(4) enat-the-enat leD order-less-imp-not-less plus-enat-simps(1)*)
have *85*: *epred(llength ys)* < *llength ys*
by (*metis 10 assms(10) co.enat.exhaust-sel enat-ord-code(4) enat-the-enat ile-eSuc iless-Suc-eq leD llength-eq-0 order-neq-le-trans*)

have *86*: *llength* (*lfuse ys* (*lmap* (λx::*nat*. *x* + *cp*) *zs*)) = *llength ys* + *epred*(*llength zs*)
by (*simp add*: *assms*(*10*) *lfuse-llength*)
have *87*: ((*epred* (*llength ys*) + *epred*(*llength zs*))) < *llength ys* + *epred*(*llength zs*)
by (*metis 83 84 85 add.commute enat-le-plus-same*(*2*) *enat-less-enat-plusI2 enat-the-enat infinity-ileE*)
have *82*: *enat* (*the-enat* (*epred* (*llength ys*)) + (*the-enat* (*epred*(*llength zs*)))) < *llength* (*lfuse ys* (*lmap* (λx::*nat*. *x* + *cp*) *zs*))
using *83 84 86 87* **by** *presburger*
have *11*: *enat* (*the-enat* (*epred* (*llength ys*)) + *i*) < *llength* (*lfuse ys* (*lmap* (λx::*nat*. *x* + *cp*) *zs*))
using *81 82 order-le-less-trans* **by** *blast*
have *12*: (*the-enat* (*epred* (*llength ys*)) + *i* − *the-enat* (*epred* (*llength ys*))) = *i*
by *auto*
have *4*: *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) = (*lnth* (*lmap* (λ*x*. *x*+*cp*) *zs*) *i*)
by (*simp add*: *1 10 11 assms*(*13*) *assms*(*9*) *lfuse-lnth-c*)
have *5*: (*lnth* (*lmap* (λ*x*. *x*+*cp*) *zs*) *i*) = *cp* + (*lnth zs i*)
by (*simp add*: *assms*)
show *?thesis*
using *4 5* **by** *presburger*
qed

lemma *lidx-lfuse-lnth-cp*:
assumes *lidx ys*
(*lnth ys 0*) = *0*
lidx zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite zs
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
¬ *lnull ys*
llast ys = *cp*
llast zs = *the-enat*(*epred* (*llength xs*)) − *cp*
i < (*llength zs*)
cp < *llength xs*
shows *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) = *cp* + (*lnth zs i*)
using *assms ridx-lidx ridx-lfuse-lnth-cp* **by** *blast*

lemma *ridx-lfuse-lnth-cp-a*:
assumes *ridx R ys*
(*lnth ys 0*) = *0*
ridx R zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite zs
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
¬ *lnull ys*

llast ys = cp
llast zs = the-enat(epred (llength xs)) − cp
i < (epred(llength ys)) + (llength zs)
the-enat(epred(llength ys)) ≤ i
cp < llength xs
shows *lnth (lfuse ys (lmap (λx. x+cp) zs)) i = cp + (lnth zs (i −the-enat(epred(llength ys))))*
proof −
have *1*: *i = (the-enat (epred (llength ys)) + (i − the-enat (epred (llength ys))))*
using *assms* **by** *fastforce*
have *2*: *enat (i − the-enat (epred (llength ys))) < llength zs*
using *assms*
by (*metis 1 enat-add-mono epred-enat lfinite-llength-enat plus-enat-simps(1) the-enat.simps*)
show *?thesis*
using *1 2 assms ridx-lfuse-lnth-cp*[*of R ys zs xs cp (i −the-enat(epred(llength ys)))*]
by *presburger*
qed

lemma *lidx-lfuse-lnth-cp-a*:
assumes *lidx ys*
(lnth ys 0) = 0
lidx zs
(lnth zs 0) = 0
lfinite ys
lfinite zs
lfinite xs
¬ lnull xs
¬ lnull zs
¬ lnull ys
llast ys = cp
llast zs = the-enat(epred (llength xs)) − cp
i < (epred(llength ys)) + (llength zs)
the-enat(epred(llength ys)) ≤ i
cp < llength xs
shows *lnth (lfuse ys (lmap (λx. x+cp) zs)) (i) = cp + (lnth zs (i −the-enat(epred(llength ys))))*
using *assms ridx-lidx ridx-lfuse-lnth-cp-a* **by** *blast*

lemma *ridx-lfuse-lnth-cp-llast*:
assumes *ridx R ys*
(lnth ys 0) = 0
ridx R zs
(lnth zs 0) = 0
lfinite ys
lfinite zs
lfinite xs
¬ lnull xs
¬ lnull zs
¬ lnull ys
llast ys = cp
llast zs = the-enat(epred (llength xs)) − cp
i < (llength zs)

```
        cp < llength xs
shows  llast (lfuse ys (lmap (λx. x+cp) zs)) = (the-enat (epred(llength xs)))
proof −
 have 1: llast (lfuse ys (lmap (λx. x+cp) zs)) = llast (lmap (λx. x+cp) zs)
   using assms
  by (metis add-cancel-left-left lfinite-lmap lfirst-def llast-lfuse llist.map-disc-iff
      llist.map-sel(1) lnth-0-conv-lhd)
 have 2: llast (lmap (λx. x+cp) zs) = cp + (llast zs)
   by (simp add: assms(6) assms(9) llast-lmap)
 have 3: cp + (llast zs) = (the-enat (epred(llength xs)))
   using assms
   by (metis add-diff-inverse-nat co.enat.exhaust-sel enat-ord-simps(2) enat-the-enat ileI1
      ile-eSuc infinity-ileE leD lfinite-conv-llength-enat llength-eq-0)
 show ?thesis using 1 2 3 by auto
qed

lemma lidx-lfuse-lnth-cp-llast:
 assumes  lidx ys
         (lnth ys 0 ) = 0
         lidx zs
        (lnth zs 0 ) = 0
        lfinite ys
        lfinite zs
        lfinite xs
        ¬ lnull xs
        ¬ lnull zs
        ¬ lnull ys
         llast ys = cp
         llast zs = the-enat(epred (llength xs)) − cp
         i < (llength zs)
         cp < llength xs
shows  llast (lfuse ys (lmap (λx. x+cp) zs)) = (the-enat (epred(llength xs)))
using assms ridx-lidx ridx-lfuse-lnth-cp-llast by blast

lemma ridx-lfuse-lnth-cp-infinite:
 assumes  ridx R ys
         (lnth ys 0 ) = (0::nat)
         ridx R zs
        (lnth zs 0 ) = 0
        lfinite ys
        ¬lfinite zs
        ¬lfinite xs
        ¬ lnull ys
         llast ys = cp
 shows lnth (lfuse ys (lmap (λx. x+cp) zs)) (the-enat(epred(llength ys)) + i) = cp + (lnth zs i)
proof −
 have 1: llast ys = lfirst(lmap (λx. x+cp) zs)
   by (metis assms(4) assms(6) assms(9) lfirst-def llist.map-sel(1) lnth-0-conv-lhd
      lnull-imp-lfinite plus-nat.add-0)
 have 3: lfirst(lmap (λx. x+cp) zs) = cp + (lnth zs 0)
```

using *1 assms* **by** *auto*
have *8*: *i*≤ *epred*(*llength zs*)
by (*metis assms*(*6*) *enat.simps*(*3*) *ldrop-eq-LNil lfinite-ldrop lfinite-ltl linorder-le-cases llength-ltl*)
have *10*: *epred* (*llength ys*) ≤ *enat* (*the-enat* (*epred* (*llength ys*)) + (*i*::*nat*))
by (*metis add.right-neutral add-le-same-cancel1 assms*(*5*) *assms*(*8*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *enat-the-enat ile-eSuc infinity-ileE le-zero-eq lfinite-llength-enat linorder-le-cases llength-eq-0*)
have *11*: *enat* (*the-enat* (*epred* (*llength ys*)) + *i*) < *llength* (*lfuse ys* (*lmap* (λ*x*::*nat*. *x* + *cp*) *zs*))
using *assms*
by (*metis 1 enat-ile ldrop-lfuse lfinite-conv-llength-enat lfinite-ldrop lfinite-lmap not-le-imp-less*)
have *12*: (*the-enat* (*epred* (*llength ys*)) + *i* − *the-enat* (*epred* (*llength ys*))) = *i*
by *auto*
have *4*: *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) =
(*lnth* (*lmap* (λ*x*. *x*+*cp*) *zs*) *i*)
using *assms*
by (*metis 1 10 11 12 lfuse-lnth-c llist.map-disc-iff lnull-imp-lfinite*)
have *5*: (*lnth* (*lmap* (λ*x*. *x*+*cp*) *zs*) *i*) = *cp* + (*lnth zs i*)
using *assms*
by (*metis 8 add.commute co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lmap lnull-imp-lfinite*)
show *?thesis*
using *4 5* **by** *presburger*
qed

lemma *lidx-lfuse-lnth-cp-infinite*:
assumes *lidx ys*
(*lnth ys 0*) = *0*
lidx zs
(*lnth zs 0*) = *0*
lfinite ys
¬*lfinite zs*
¬*lfinite xs*
¬ *lnull ys*
llast ys = *cp*
shows *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) = *cp* + (*lnth zs i*)
using *assms ridx-lidx ridx-lfuse-lnth-cp-infinite* **by** *blast*

lemma *lidx-lfuse-lidx*:
assumes *lidx ys*
lnth ys 0 = *0*
lidx zs
lnth zs 0 = *0*
lfinite ys
¬ *lnull ys*
llast ys = *cp*
lfinite zs
¬ *lnull zs*
lfinite xs
llast zs = *the-enat*(*epred*(*llength xs*)) −*cp*

```
        cp < llength xs
        i < llength zs
        cp < llength xs
 shows  lidx (lfuse ys (lmap (λx. x+ cp) zs)) ∧ (lnth (lfuse ys (lmap (λx. x+ cp) zs)) 0) = 0
proof −
 have 1: llast ys = lfirst(lmap (λx. x+ cp) zs)
   by (metis assms(4) assms(7) assms(9) cancel-comm-monoid-add-class.diff-cancel diff-add
       dual-order.refl lfirst-def llist.map-sel(1) lnth-0-conv-lhd)
 have 2: lfirst (lfuse ys (lmap (λx. x+ cp) zs)) = lfirst ys
   by (simp add: assms(6) assms(9) lfirst-lfuse-1)
 have 3: llength (lfuse ys (lmap (λx. x+ cp) zs)) = llength ys + epred(llength zs)
  by (simp add: assms(13) assms(6) lfuse-llength)
 have 4: ⋀j. j<llength ys ⟹ lnth (lfuse ys (lmap (λx. x+ cp) zs)) j= lnth ys j
   using lfuse-lnth-a by blast
 have 40: ∃ k1. llength ys = (enat k1)
   by (simp add: assms(5) lfinite-llength-enat)
 obtain k1 where 41: llength ys = (enat k1)
   using 40 by blast
 have 42: (enat (k1 −1)) = epred(llength ys)
     using 41 epred-enat by presburger
 have 43: 0<k1
   using assms 41
   by (metis gr0I llength-eq-0 zero-enat-def)
 have 44: the-enat(epred(llength ys)) = (k1 −1)
   by (metis 42 the-enat.simps)
 have 5: ⋀j.  epred(llength ys) ≤j ∧ j < epred(llength ys) + llength zs ⟹
         lnth (lfuse ys (lmap (λx. x+ cp) zs)) j =
         cp + (lnth zs (j− the-enat(epred(llength ys))))
    using assms lidx-lfuse-lnth-cp-a[of ys zs xs cp]
     by (metis  enat-ord-simps(1) enat-the-enat gr-implies-not-zero
         infinity-ileE  llength-eq-0)
 have 45: ⋀j.  k1−1 ≤j ∧ j < (enat (k1−1)) + llength zs ⟹
         lnth (lfuse ys (lmap (λx. x+ cp) zs)) j =
         cp + (lnth zs (j− (k1−1)))
    using 5 44 by (metis 42 enat-ord-simps(1))
 have 50: llength(lfuse ys (lmap (λx. x+ cp) zs)) = epred(llength ys) + llength zs
   using assms
   by (metis 3 add.commute epred-iadd1 llength-eq-0)
 have 51: ⋀j. enat (Suc j) < llength ys ⟹
              (lnth (lfuse ys (lmap (λx. x+ cp) zs)) j) <
              (lnth (lfuse ys (lmap (λx. x+ cp) zs)) (Suc j))
     by (metis assms(1) assms(5) eSuc-enat iless-Suc-eq lfinite-conv-llength-enat lfuse-lnth-a
         lidx-def not-le-imp-less not-less-iff-gr-or-eq)
 have 52: ⋀j.  k1−1 ≤j ∧ (Suc j) < enat (k1−1) + llength zs ⟹
                cp + (lnth zs (j− (k1−1))) <
                cp + (lnth zs ((Suc j)− (k1−1)))
        using assms(3) unfolding lidx-def
        by simp
         (metis Suc-diff-le add-Suc-right assms(8) enat-ord-simps(2) leD lfinite-llength-enat
          nat-add-left-cancel-le not-le-imp-less ordered-cancel-comm-monoid-diff-class.add-diff-inverse
```

```
        plus-enat-simps(1))
 have 6: ⋀j. enat (Suc j) < llength(lfuse ys (lmap (λx. x+ cp) zs)) ⟹
              (lnth (lfuse ys (lmap (λx. x+ cp) zs)) j) <
              (lnth (lfuse ys (lmap (λx. x+ cp) zs)) (Suc j))
    proof −
      fix j
      assume a: enat (Suc j) < llength(lfuse ys (lmap (λx. x+ cp) zs))
      show (lnth (lfuse ys (lmap (λx. x+ cp) zs)) j) < (lnth (lfuse ys (lmap (λx. x+ cp) zs)) (Suc j))
      proof −
       have 61: enat (Suc j) < llength ys ⟹
            (lnth (lfuse ys (lmap (λx. x+ cp) zs)) j) < (lnth (lfuse ys (lmap (λx. x+ cp) zs)) (Suc j))
         using 51 by blast
       have 62: k1−1 ≤j ∧ (Suc j) < llength(lfuse ys (lmap (λx. x+ cp) zs)) ⟹
            (lnth (lfuse ys (lmap (λx. x+ cp) zs)) j) < (lnth (lfuse ys (lmap (λx. x+ cp) zs)) (Suc j))
         by (metis 42 45 50 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le)
       show ?thesis
       by (metis 41 43 61 62 Suc-diff-1 Suc-mono a enat-ord-simps(2) leI)
      qed
    qed
 show ?thesis unfolding lidx-def
 by (simp add: 41 43 6 assms(13) assms(2) lfuse-lnth-a)
qed

lemma lidx-lfuse-lidx-infinite:
 assumes lidx  ys
        lnth ys 0 = 0
        lidx  zs
        lnth zs 0 = 0
        lfinite ys
        ¬ lnull ys
        llast ys  = cp
        ¬lfinite zs
        ¬lfinite xs
 shows   lidx (lfuse ys (lmap (λx. x+ cp) zs)) ∧ (lnth (lfuse ys (lmap (λx. x+ cp) zs)) 0) = 0
proof −
 have 1: llast ys = lfirst(lmap (λx. x+ cp) zs)
    by (metis assms(4) assms(7) assms(8) lfirst-def llist.map-sel(1) lnth-0-conv-lhd
        lnull-imp-lfinite plus-nat.add-0)
 have 2: lfirst (lfuse ys (lmap (λx. x+ cp) zs)) = lfirst ys
   by (simp add: assms(6) lfirst-lfuse-1)
 have 4: ⋀j. j<llength ys ⟹ lnth (lfuse ys (lmap (λx. x+ cp) zs)) j= lnth ys j
   using lfuse-lnth-a by blast
 have 40: ∃ k1. llength ys = (enat k1)
   by (simp add: assms(5) lfinite-llength-enat)
 obtain k1 where 41: llength ys = (enat k1)
   using 40 by blast
 have 42: (enat (k1 −1)) = epred(llength ys)
     using 41 epred-enat by presburger
 have 43: 0<k1
   using assms 41
```

by (*metis gr0I llength-eq-0 zero-enat-def*)
have *44*: *the-enat*(*epred*(*llength ys*)) = (*k1* −*1*)
by (*metis 42 the-enat.simps*)
have *5*: ⋀*j*. *epred*(*llength ys*) ≤*j* ∧ *j* < *epred*(*llength ys*) + *llength zs* ⟹
lnth (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j* =
cp + (*lnth zs* (*j*− *the-enat*(*epred*(*llength ys*))))
using *assms lidx-lfuse-lnth-cp-infinite*[*of ys zs xs cp*]
by (*metis 42 44 enat-ord-simps*(*1*) *ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *45*: ⋀*j*. *k1*−*1* ≤*j* ∧ *j* < (*enat* (*k1*−*1*)) + *llength zs* ⟹
lnth (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j* =
cp + (*lnth zs* (*j*− (*k1*−*1*)))
using *5 44* **by** (*metis 42 enat-ord-simps*(*1*))
have *46*: *llength ys* + *epred* (*llength zs*) = *epred* (*llength ys*) + *llength zs*
by (*metis add.commute assms*(*6*) *assms*(*8*) *epred-iadd1 llength-eq-0 lnull-imp-lfinite*)
have *50*: *llength*(*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) = *epred*(*llength ys*) + *llength zs*
using *lfuse-llength*[*of ys* (*lmap* (λ*x*::*nat*. *x* + *cp*) *zs*)]
llength-lmap[*of* (λ*x*::*nat*. *x* + *cp*) *zs*]
using *46 assms*(*6*) **by** *presburger*
have *51*: ⋀*j*. *enat* (*Suc j*) < *llength ys* ⟹
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j*) <
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*))
by (*metis assms*(*1*) *assms*(*5*) *eSuc-enat iless-Suc-eq lfinite-conv-llength-enat lfuse-lnth-a*
lidx-def not-le-imp-less not-less-iff-gr-or-eq)
have *52*: ⋀*j*. *k1*−*1* ≤*j* ∧ (*Suc j*) < *enat* (*k1*−*1*) + *llength zs* ⟹
cp + (*lnth zs* (*j*− (*k1*−*1*))) <
cp + (*lnth zs* ((*Suc j*)− (*k1*−*1*)))
using *assms*(*3*) **unfolding** *lidx-def* **by** *simp*
(*metis Nat.add-diff-assoc assms*(*8*) *enat-ile lfinite-conv-llength-enat not-le-imp-less*
plus-1-eq-Suc)
have *53*: ⋀*j*. *k1*−*1* ≤*j* ∧ (*Suc j*) < *enat* (*k1*−*1*) + *llength zs* ⟹
lnth (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j* <
lnth (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*)
by (*metis 45 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le*)
have *6*: ⋀*j*. *enat* (*Suc j*) < *llength*(*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) ⟹
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j*) <
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*))
by (*metis 41 42 43 50 51 53 Suc-diff-1 Suc-eq-plus1 add-less-cancel-right*
enat-ord-simps(*2*) *leI*)
show *?thesis* **unfolding** *lidx-def*
by (*simp add*: *41 43 6 assms lfuse-lnth-a*)
qed

1.1.5 lsub

lemma *lsub-eq-lsubc*:
assumes *k*≤ *n*
n < *llength xs*
shows *lsub k n xs* = *lsubc k n xs*
using *assms*
unfolding *lsub-def lsubc-def*

by (*auto simp add*: *min-def*)
(*metis co.enat.exhaust-sel enat-ord-simps*(*1*) *iless-Suc-eq ldrop-lnull llength-eq-0*
order-le-less-trans)

lemma *lsub-same*:
assumes (*enat k*)<*llength xs*
shows *lsub k k xs* = (*LCons* (*lnth xs k*) *LNil*)
using *assms*
unfolding *lsub-def*
by (*metis LNil-eq-ltake-iff diff-is-0-eq*′ *enat-0-iff*(*1*) *ldrop-enat ldropn-Suc-conv-ldropn*
ltake-eSuc-LCons order.order-iff-strict)

lemma *lsubc-same*:
assumes (*enat k*)<*llength xs*
shows *lsubc k k xs* = (*LCons* (*lnth xs k*) *LNil*)
using *assms*
lsub-eq-lsubc[*of k k xs*] *lsub-same*[*of k xs*]
by *fastforce*

lemma *llength-lsub*:
assumes *k*≤ *n*
n < *llength xs*
shows *llength* (*lsub k n xs*) = (*eSuc* (*n*−*k*))
proof −
have *1*: *llength* (*ldrop* (*enat k*) *xs*) = (*llength xs* − *enat k*)
by (*simp add*: *ldrop-enat*)
have *2*: *min* (*eSuc* (*enat* (*n* − *k*))) (*llength xs* − *enat k*) = (*eSuc* (*n*−*k*))
by (*metis 1 Suc-diff-le assms*(*1*) *assms*(*2*) *eSuc-enat enat-llength-ldropn ileI1 ldrop-enat min.absorb1*)
have *3*: *llength* (*ltake* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) *xs*)) = (*eSuc* (*n*−*k*))
by (*simp add*: *1 2*)
show *?thesis*
by (*metis 3 idiff-enat-enat lsub-def*)
qed

lemma *llength-lsubc*:
assumes *k*≤ *n*
n < *llength xs*
shows *llength* (*lsubc k n xs*) = (*eSuc* (*n*−*k*))
using *assms lsub-eq-lsubc*[*of k n xs*] *llength-lsub*[*of k n xs*] **by** *presburger*

lemma *llength-lsub-a*:
shows *llength* (*lsub k n xs*) =
min (*eSuc* (*n* − *k*))
(*if k* = ∞ *then 0*::*enat else llength xs* − *k*)
unfolding *lsub-def*
using *llength-ltake*[*of eSuc* (*n*−*k*) (*ldrop k xs*)] *llength-ldrop*[*of k xs*]
using *idiff-enat-enat* **by** *presburger*

lemma *llength-lsubc-a*:
shows *llength* (*lsubc k n xs*) =

min (*eSuc* (*n* − *k*))
(*if min k* (*epred* (*llength xs*)) = ∞
then 0::*enat*
else llength xs − *min k* (*epred* (*llength xs*)))
unfolding *lsubc-def*
using *llength-ldrop*[*of* (*min k* (*epred* (*llength xs*))) *xs*]
using *llength-ltake*[*of* (*eSuc* (*n* − *k*)) (*ldrop* (*min k* (*epred* (*llength xs*))) *xs*)]
using *idiff-enat-enat* **by** *presburger*

lemma *lsub-not-lnull*:
assumes *k*≤ *n*
n < *llength xs*
shows ¬*lnull* (*lsub k n xs*)
using *assms*
by (*metis eSuc-enat idiff-enat-enat llength-LNil llength-lsub llist.collapse*(*1*) *zero-ne-eSuc*)

lemma *lsub-llength-gr-one*:
assumes *k*<*n*
n < *llength xs*
shows *1*< *llength* (*lsub k n xs*)
using *llength-lsub-a*[*of k n xs*] *assms*
by (*auto simp add*: *min-def one-eSuc zero-enat-def llength-lsub*)

lemma *lsub-lfinite*:
assumes *k*≤ *n*
n < *llength xs*
shows *lfinite* (*lsub k n xs*)
using *assms*
by (*simp add*: *eSuc-enat llength-eq-enat-lfiniteD llength-lsub*)

lemma *lnth-lsub*:
assumes *n* < *llength xs*
k+*j* ≤ *n*
shows *lnth* (*lsub k n xs*) *j* = (*lnth xs* (*k*+*j*))
proof −
have *1*: *enat* (*j*::*nat*) < *eSuc* (*enat* ((*n*::*nat*) − (*k*::*nat*)))
using *assms*(*2*) **by** *auto*
have *2*: *lnth* (*ltake* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) (*xs*::′*a llist*))) *j* =
lnth (*ldrop* (*enat k*) *xs*) *j*
using *1 lnth-ltake* **by** *blast*
have *3*: *lnth* (*ldrop* (*enat k*) *xs*) *j* = (*lnth xs* (*k*+*j*))
by (*metis add.commute assms*(*1*) *assms*(*2*) *enat-ord-simps*(*1*) *ldrop-enat lnth-ldropn order-le-less-trans*)
show *?thesis* **unfolding** *lsub-def*
using *2 3* **by** *presburger*
qed

lemma *lnth-lsub-a*:
assumes *n* < *llength xs*
m ≤ *n*

$k \leq m$
shows *lnth* (*lsub k n xs*) (*m*−*k*) = (*lnth xs m*)
using *assms* **by** (*simp add*: *lnth-lsub*)

lemma *ltake-lsub*:
assumes $n <$ *llength xs*
$m + k \leq n$
shows *ltake* (*eSuc m*) (*lsub k n xs*) = *lsub k* (*m*+*k*) *xs*
proof −
have *1*: *ltake* (*eSuc m*) (*ltake* (*eSuc* (*n* − *k*)) (*ldrop k xs*)) =
ltake (*eSuc m*) (*ldrop k xs*)
using *ltake-ltake*[*of* (*eSuc m*) (*eSuc* (*n* − *k*)) (*ldrop k xs*)]
using *Nat.le-diff-conv2 assms*(*2*) **by** *auto*
have *2*: *ltake* (*eSuc* (*m* + *k* − *k*)) (*ldrop k xs*) = *ltake* (*eSuc m*) (*ldrop k xs*)
by *simp*
show *?thesis*
unfolding *lsub-def* **using** *1 2* **by** *presburger*
qed

lemma *ldrop-lsub*:
assumes $n <$ *llength xs*
$m + k \leq n$
shows *ldrop m* (*lsub k n xs*) = *lsub* (*m*+*k*) *n xs*
proof −
have *1*: (*ltake* (*eSuc* (*n* − *k*)) (*ldrop k xs*)) =
ldrop k (*ltake* (*eSuc n*) *xs*)
by (*metis Suc-diff-le add-leD2 assms*(*2*) *eSuc-enat idiff-enat-enat ldrop-ltake*)
have *2*: *ldrop m* (*ldrop k* (*ltake* (*eSuc n*) *xs*)) =
ldrop (*m* +*k*) (*ltake* (*eSuc n*) *xs*)
by *simp*
show *?thesis* **unfolding** *lsub-def* **using** *1 2*
by (*simp add*: *Suc-diff-le assms*(*2*) *eSuc-enat ldrop-ltake*)
qed

lemma *ltl-lsub*:
assumes $n <$ *llength xs*
$k \leq n$
shows *ltl*(*lsub k n xs*) = (*if k*=*n then LNil else lsub k* (*n*−*1*) (*ltl xs*))
proof −
have *1*: (*lsub k n xs*) = (*ltake* (*eSuc* (*n* − *k*)) (*ldrop k xs*))
unfolding *lsub-def* **by** *simp*
have *2*: $k{=}n \Longrightarrow$ *?thesis*
by (*simp add*: *assms*(*1*) *lsub-same*)
have *25*: $k \neq n \Longrightarrow$ (*epred* (*eSuc* (*n* − *k*))) = (*eSuc* (*enat* (*n* − (*1*::*nat*) − *k*)))
by (*metis Suc-diff-1 assms*(*2*) *co.enat.sel*(*2*) *diff-commute eSuc-enat order-neq-le-trans zero-less-diff*)
have *3*: $k \neq n \Longrightarrow$ *?thesis*
using *assms ltl-ltake*[*of* (*eSuc* (*n* − *k*)) (*ldrop k xs*)]
ltl-ldrop[*of k xs*]
unfolding *lsub-def*
using *25* **by** *presburger*

show *?thesis*
using *2 3* **by** *blast*
qed

lemma *ltl-ldrop-one*:
ltl xs = ldrop 1 xs
by (*metis ldrop-0 ldrop-ltl one-eSuc*)

lemma *lappend-lsub-ltl-lsub*:
assumes *k ≤ n*
n ≤ m
m < llength xs
shows *lappend* (*lsub k n xs*) (*ltl* (*lsub n m xs*)) = *lsub k m xs*
proof −
have *0*: (*ltl* (*lsub n m xs*)) = *ldrop 1* (*lsub n m xs*)
by (*metis ldrop-0 ldrop-ltl one-eSuc*)
have *1*: (*lsub k n xs*) = (*ltake* (*eSuc* (*n* − *k*)) (*ldrop k xs*))
unfolding *lsub-def* **by** *simp*
have *2*: (*lsub n m xs*) = (*ltake* (*eSuc* (*m* − *n*)) (*ldrop n xs*))
unfolding *lsub-def* **by** *simp*
have *3*: *lsub k m xs* = (*ltake* (*eSuc* (*m* − *k*)) (*ldrop k xs*))
unfolding *lsub-def* **by** *simp*
have *8*: *ltake* (*eSuc* (*enat* ((*n*::*nat*) − (*k*::*nat*))) + *enat* ((*m*::*nat*) − *n*)) (*ldrop* (*enat k*) (*xs*::′*a llist*)) =
(*ltake* (*eSuc* (*m* − *k*)) (*ldrop k xs*))
by (*simp add*: *assms*(*1*) *assms*(*2*) *eSuc-enat*)
have *9*: (*ltake* (*enat* (*m* − *n*)) (*ldrop* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) *xs*))) =
ltl(*ltake* (*eSuc* (*m* − *n*)) (*ldrop n xs*))
by (*metis 0 2 add.commute assms*(*1*) *eSuc-minus-1 ldrop-ldrop ldrop-ltake le-add-diff-inverse*
plus-1-eSuc(*1*) *plus-enat-simps*(*1*))
have *10*: *ltake* (*eSuc* (*enat* ((*n*::*nat*) − (*k*::*nat*))) + *enat* ((*m*::*nat*) − *n*)) (*ldrop* (*enat k*) (*xs*::′*a llist*)) =
lappend (*ltake* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) *xs*))
(*ltake* (*enat* (*m* − *n*)) (*ldrop* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) *xs*)))
using *ltake-plus-conv-lappend*[*of* (*eSuc* (*n* − *k*)) *m*−*n* (*ldrop k xs*)] **by** *blast*
show *?thesis*
using *1 10 2 3 8 9* **by** *fastforce*
qed

lemma *lsub-lfuse*:
assumes
k ≤ n
n ≤ m
m < llength xs
shows *lfuse* (*lsub k n xs*) (*lsub n m xs*) = *lsub k m xs*
using *assms*
by (*simp add*: *lappend-lsub-ltl-lsub lfuse-def lsub-not-lnull order-le-less-subst2*)

lemma *llast-lsub*:
assumes *lfinite xs*
¬ *lnull xs*
k ≤ n

$n < llength\ xs$
shows $llast\ (lsub\ k\ n\ xs) = (lnth\ xs\ n)$
using *assms*
using *lnth-lsub*[*of n xs k*]
by (*metis enat-ord-simps*(*1*) *idiff-enat-enat llast-conv-lnth llength-lsub not-le-imp-less order-less-irrefl ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)

lemma *lfirst-lsub*:
assumes $\neg\ lnull\ xs$
$k \leq n$
$n < llength\ xs$
shows $lfirst\ (lsub\ k\ n\ xs) = (lnth\ xs\ k)$
using *assms*
by (*simp add*: *ldrop-enat lfirst-def lhd-ldropn lsub-def order-le-less-subst2*)

lemma *lsub-lfuse-lidx*:
assumes *lidx ls*
lfinite ls
lfinite xs
$\neg\ lnull\ xs$
$llast\ ls = epred(llength\ xs)$
$(Suc\ i) < (llength\ ls)$
shows $lfuse\ (lsub\ (lnth\ ls\ i)\ (lnth\ ls\ (Suc\ i))\ xs)\ (lsub\ (lnth\ ls\ (Suc\ i))\ (llast\ ls)\ xs) = (lsub\ (lnth\ ls\ i)\ (llast\ ls)\ xs)$
proof −
have *1*: $(lnth\ ls\ i) \leq (lnth\ ls\ (Suc\ i))$
by (*simp add*: *assms*(*1*) *assms*(*6*) *lidx-less-eq*)
have *2*: $llast\ ls = (lnth\ ls\ (the\text{-}enat(epred\ (llength\ ls))))$
using *llast-conv-lnth*
by (*metis assms*(*2*) *assms*(*6*) *co.enat.collapse enat.simps*(*3*) *enat-ord-simps*(*4*) *enat-the-enat gr-implies-not-zero ile-eSuc lfinite-llength-enat not-less-iff-gr-or-eq order-neq-le-trans*)
have *3*: $1 < llength\ ls$
by (*metis assms*(*1*) *assms*(*6*) *enat-ord-simps*(*1*) *gr-implies-not-zero leD le-add1 lidx-expand-1 llength-eq-0 one-enat-def plus-1-eq-Suc*)
have *4*: $(lnth\ ls\ (Suc\ i)) \leq (llast\ ls)$
using *2 assms*(*1*) *assms*(*2*) *assms*(*6*) *lfinite-llength-enat lidx-less-eq* **by** *fastforce*
have *5*: $enat\ (lnth\ ls\ (Suc\ i)) < llength\ xs$
by (*metis 4 assms*(*4*) *assms*(*5*) *co.enat.exhaust-sel eSuc-enat enat-ord-simps*(*2*) *le-imp-less-Suc llength-eq-0*)
have *6*: $llast\ (lsub\ (lnth\ ls\ i)\ (lnth\ ls\ (Suc\ i))\ xs) = (lnth\ xs\ (lnth\ ls\ (Suc\ i)))$
using *llast-lsub*[*of xs* $(lnth\ ls\ i)$ $(lnth\ ls\ (Suc\ i))$]
using *1 5 assms*(*3*) *assms*(*4*) *enat-ord-simps*(*1*) **by** *blast*
have *7*: $lfirst\ (lsub\ (lnth\ ls\ (Suc\ i))\ (llast\ ls)\ xs) = (lnth\ xs\ (lnth\ ls\ (Suc\ i)))$
by (*metis 4 assms*(*4*) *assms*(*5*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *ile-eSuc iless-Suc-eq lfirst-lsub llength-eq-0 order-less-le*)
show *?thesis*
by (*metis 1 4 assms*(*4*) *assms*(*5*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *ile-eSuc iless-Suc-eq llength-eq-0 lsub-lfuse order-neq-le-trans*)
qed

1.1.6 llastlfirst

lemma *llastlfirst-ridx*:
llastlfirst xss = ridx (*λ a b. llast a = lfirst b*) *xss*
by (*simp add*: *llastlfirst-def ridx-def*)

lemma *llastlfirst-LNil*[*simp*]:
llastlfirst LNil
unfolding *llastlfirst-def* **by** *simp*

lemma *llastlfirst-LOne*[*simp*]:
llastlfirst (*LCons xs LNil*)
unfolding *llastlfirst-def* **by** (*simp add*: *zero-enat-def*)

lemma *llastlfirst-LCons*[*simp*]:
assumes ¬*lnull xss*
shows *llastlfirst* (*LCons xs xss*) ⟷ *llast xs = lfirst* (*lfirst xss*) ∧ *llastlfirst xss*
using *assms* **unfolding** *llastlfirst-def*
by *auto*
(*metis One-nat-def lfirst-def lhd-LCons-ltl lnth-LCons′ not-lnull-llength one-enat-def*,
metis Suc-ile-eq lnth-Suc-LCons,
metis One-nat-def Suc-diff-le Suc-ile-eq add-diff-cancel-left′ le-Suc-eq le-add1 lfirst-def
llist.disc(*2*) *lnth-0 lnth-0-conv-lhd lnth-ltl ltl-simps*(*2*) *plus-1-eq-Suc*)

lemma *llastlfirst-lappend-lfinite*:
assumes *lfinite xss*
shows *llastlfirst* (*lappend xss yss*) ⟷
(*llastlfirst xss* ∧ *llastlfirst yss* ∧
(*if lnull xss* ∨ *lnull yss then True else llast*(*llast xss*) = *lfirst*(*lfirst yss*)))
using *assms*
by (*metis* (*mono-tags*, *lifting*) *lfirst-def llastlfirst-ridx ridx-lappend-lfinite*)

1.1.7 lfusecat

context notes [[*function-internals*]]
begin

partial-function (*llist*) *lfusecat* :: *′a llist llist* ⇒ *′a llist*
where *lfusecat xss* = (*case xss of LNil* ⇒ *LNil* | *LCons xs xss′* ⇒ *lfuse xs* (*lfusecat xss′*))

end

lemma *lfusecat-simps* [*simp*, *code*]:
shows *lfusecat-LNil*: *lfusecat LNil = LNil*
and *lfusecat-LCons*: *lfusecat* (*LCons xs xss*) = *lfuse xs* (*lfusecat xss*)
by (*simp-all add*: *lfusecat.simps*)

declare *lfusecat.mono*[*cont-intro*]

lemma *mono2mono-lfusecat*[*THEN llist.mono2mono*, *cont-intro*, *simp*]:
shows *monotone-lfusecat*: *monotone* (*lprefix*) (*lprefix*) *lfusecat*

by(*rule llist.fixp-preserves-mono1*[*OF lfusecat.mono lfusecat-def*]) *simp*

lemma *mcont2mcont-lfusecat*[*THEN llist.mcont2mcont*, *cont-intro*, *simp*]:
shows *mcont-lfusecat*: *mcont lSup* (*lprefix*) *lSup* (*lprefix*) *lfusecat*
by (*rule llist.fixp-preserves-mcont1*[*OF lfusecat.mono lfusecat-def*])
simp

lemmas *lfusecat-fixp-parallel-induct* =
parallel-fixp-induct-1-1[*OF llist-partial-function-definitions llist-partial-function-definitions*
lfusecat.mono lfusecat.mono lfusecat-def lfusecat-def, *case-names adm LNil step*]

lemma *llist-all2-lfusecatI*:
llist-all2 (*llist-all2 A*) *xss yss*
$\Longrightarrow$ *llist-all2 A* (*lfusecat xss*) (*lfusecat yss*)
proof(*induct arbitrary*: *xss yss rule*: *lfusecat-fixp-parallel-induct*)
case *adm*
then show *?case* **by** (*auto split*: *llist.split intro*: *llist-all2-lappendI*)
next
case *LNil*
then show *?case* **by** (*auto split*: *llist.split intro*: *llist-all2-lappendI*)
next
case (*step f g*)
then show *?case*
using *llist-all2-lfuseI* **by** (*auto split*: *llist.split intro*: *llist-all2-lappendI*) *blast*
qed

lemma *not-lnull-lset-conv-a*:
$\neg$ *lnull xss* $\wedge$ ($\forall$ *xs* $\in$ *lset xss*. $\neg$ *lnull xs*) $\longleftrightarrow$ *lset xss* $\neq$ {} $\wedge$ *LNil* $\notin$ *lset xss*
using *llist.collapse*(*1*) *llist.disc*(*1*) *lset-eq-empty* **by** *blast*

lemma *not-lnull-lset-conv-b*:
$\neg$ *lnull xss* $\wedge$ ($\forall$ *xs* $\in$ *lset xss*. $\neg$ *lnull xs*) $\longleftrightarrow$ *lset xss* $\neq$ {} $\wedge$ *lset xss* $\cap$ {*LNil*} = {}
using *llist.collapse*(*1*) **by** *force*

lemma *lfusecat-eq-LNil*:
lfusecat xss = *LNil* $\longleftrightarrow$ *lset xss* $\subseteq$ {*LNil*}
proof (*induction xss*)
case *adm*
then show *?case*
by *simp*
next
case *LNil*
then show *?case*
by *simp*
next
case (*LCons xs xss*)
then show *?case*

by *simp*
(*metis* (*no-types*, *lifting*) *lfuse-conv-lnull llist.disc*(*1*) *llist.expand*)
qed

lemma *lfusecat-lfilter-neq-LNil*:
lfusecat (*lfilter* (λ*xs*. ¬ *lnull xs*) *xss*) = *lfusecat xss*
proof (*induct xss*)
case *adm*
then show *?case* **by** *simp*
next
case *LNil*
then show *?case* **by** *simp*
next
case (*LCons xs xss*)
then show *?case* **by** (*simp add*: *lappend-lnull1 lfuse-def*)
qed

lemma *lprefix-lfusecatI*:
lprefix xss yss ⟹*lprefix* (*lfusecat xss*) (*lfusecat yss*)
by (*meson mcont-lfusecat mcont-monoD*)

lemma *lset-lltl-llength*:
(∀ *xs* ∈ *lset xss*. *llength xs* ≤ *1*) ⟷ (∀ *xs* ∈ *lset* (*lltl xss*). *lnull xs*)
unfolding *lltl-def*
by (*auto simp add*: *ltl-ldrop-one*)

lemma *lset-lltl-llength-var*:
(∀ *xs* ∈ *lset xss*. *llength xs* ≤ *1*) ⟷ *lset*(*lltl xss*) ⊆ {*LNil*}
using *lset-lltl-llength*
by (*metis all-not-in-conv insert-iff lnull-def subsetI subset-singletonD*)

lemma *lset-lltl-llength-var2*:
(∀ *xs* ∈ *lset xss*. *llength xs* > *1*) ⟹ *lset* (*lltl xss*) ∩ {*LNil*} = {}
unfolding *lltl-def*
by *auto*
(*metis co.enat.exhaust-sel epred-llength less-numeral-extra*(*4*) *llength-LNil not-one-less-zero*
one-eSuc)

lemma *lfusecat-all-empty-or-LNil-a*:
assumes *lset*(*lltl xss*) ⊆ {*LNil*}
shows *llength* (*lfusecat xss*) ≤ *1*
using *assms*
proof (*induction xss*)
case *adm*
then show *?case* **unfolding** *lltl-def* **by** *simp*
next
case *LNil*

then show *?case* **by** *simp*
next
case (*LCons xs xss*)
then show *?case*
unfolding *lltl-def*
by *simp*
(*metis LCons.prems lfuse-llength-atmost-one llist.set-intros*(*1*) *lset-lltl-llength-var*)
qed

lemma *lfusecat-all-empty-or-LNil-b*:
assumes *llength* (*lfusecat xss*) ≤ *1*
shows *lset*(*lltl xss*) ⊆ {*LNil*}
using *assms*
proof (*induction xss*)
case *adm*
then show *?case* **unfolding** *lltl-def* **by** *simp*
next
case *LNil*
then show *?case* **unfolding** *lltl-def* **by** *simp*
next
case (*LCons xs xss*)
then show *?case*
unfolding *lltl-def*
by (*simp add*: *lfuse-llength-atmost-one ltl-ldrop-one*)
qed

lemma *lfusecat-all-empty-or-LNil*:
shows *llength* (*lfusecat xss*) ≤ *1* ⟷ *lset*(*lltl xss*) ⊆ {*LNil*}
using *lfusecat-all-empty-or-LNil-a lfusecat-all-empty-or-LNil-b* **by** *blast*

lemma *lfusecat-not-lnull*:
¬*lnull* (*lfusecat xss*) ⟷ ¬*lnull xss* ∧ (∃ *xs* ∈ *lset* (*xss*). ¬*lnull*(*xs*))
by (*metis lconcat-eq-LNil lfusecat-LNil lfusecat-eq-LNil lnull-def lnull-lconcat mem-Collect-eq subsetD subsetI*)

lemma *lset-eq-forall-lnull*:
lnull xss ∨ (∀ *xs* ∈ *lset* (*xss*). *lnull*(*xs*)) ⟷ *lset xss* ⊆ {*LNil*}
by (*auto simp add*: *lset-lnull*)

lemma *lfusecat-not-lnull-var*:
assumes ¬*lnull xss*
∀ *xs* ∈ *lset xss*. ¬*lnull xs*
shows ¬*lnull*(*lfusecat xss*)
using *assms*
using *lfusecat-not-lnull llist.set-sel*(*1*) **by** *blast*

lemma *lfirst-lfusecat-lfirst*:
assumes ¬*lnull xss*
¬*lnull* (*lfirst xss*)

shows *lfirst*(*lfusecat xss*) = *lfirst*(*lfirst xss*)
proof (*cases xss*)
case *LNil*
then show *?thesis*
using *assms*(*1*) *llist.disc*(*1*) **by** *blast*
next
case (*LCons x21 x22*)
then show *?thesis*
by (*metis assms*(*2*) *eq-LConsD lfirst-def lfirst-lfuse-1 lfusecat-LCons*)
qed

lemma *lfirst-lfusecat*:
assumes ¬ *lnull xs*
shows *lfirst*(*lfusecat* (*LCons xs xss*)) = *lfirst xs*
using *assms* **by** (*simp add*: *lfirst-def*)

lemma *ltl-lfusecat* :
assumes ¬*lnull xss*
¬*lnull* (*lfirst xss*)
shows *ltl*(*lfusecat xss*) = *lappend* (*ltl* (*lhd xss*)) (*ltl* (*lfusecat* (*ltl xss*)))
using *assms*
unfolding *lfirst-def*
by (*metis lappend-lnull2 lfusecat-LCons lhd-LCons-ltl ltl-lfuse*)

lemma *lfusecat-lappend*:
assumes *lfinite xss*
shows *lfusecat* (*lappend xss yss*) = *lfuse* (*lfusecat xss*) (*lfusecat yss*)
using *assms*
proof (*induct xss arbitrary*: *yss*)
case *lfinite-LNil*
then show *?case* **by** *auto*
next
case (*lfinite-LConsI xss xs*)
then show *?case*
proof −
have *1*: *lfusecat* (*lappend* (*LCons xs xss*) *yss*) = *lfuse xs* (*lfusecat* (*lappend xss yss*))
by *simp*
have *2*: *lfinite xss*
by (*simp add*: *lfinite-LConsI.hyps*(*1*))
have *3*: (*lfusecat* (*lappend xss yss*)) = *lfuse* (*lfusecat xss*) (*lfusecat yss*)
by (*simp add*: *lfinite-LConsI.hyps*(*2*))
have *4*: *lfuse xs* (*lfuse* (*lfusecat xss*) (*lfusecat yss*)) =
lfuse (*lfusecat* (*LCons xs xss*)) (*lfusecat yss*)
by (*metis lappend-lnull1 lfuse-LNil-2 lfuse-assoc lfuse-def lfusecat-LCons*)
show *?thesis*
by (*simp add*: *3 4*)
qed
qed

lemma *lfusecat-split*:
shows *lfusecat xss* = *lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))
proof −
have *1*: *xss* = *lappend* (*ltake n xss*) (*ldrop n xss*)
by (*simp add*: *lappend-ltake-ldrop*)
show *?thesis* **using** *1 lfusecat-lappend*[*of* (*ltake n xss*) (*ldrop n xss*)] **by** *force*
qed

lemma *lfuse-lfinite*:
shows *lfinite* (*lfuse xs ys*) ⟷ *lfinite xs* ∧ *lfinite ys*
by (*metis lfinite-lappend lfinite-ltl lnull-imp-lfinite ltl-lfuse*)

lemma *lfusecat-lfinite-a*:
assumes *lfinite xss*
∀ *xs* ∈ *lset xss*. *lfinite xs*
shows *lfinite* (*lfusecat xss*)
using *assms*
proof (*induct xss*)
case *lfinite-LNil*
then show *?case* **by** *auto*
next
case (*lfinite-LConsI xss xs*)
then show *?case*
proof (*cases xss*=*LNil*)
case *True*
then show *?thesis* **by** (*simp add*: *lfinite-LConsI.prems*)
next
case *False*
then show *?thesis*
proof −
have *1*: *lfinite xs*
by (*simp add*: *lfinite-LConsI.prems*)
have *5*: ∀ *xs* ∈ *lset xss*. *lfinite xs*
by (*simp add*: *lfinite-LConsI.prems*)
have *6*: *lfinite* (*lfusecat xss*)
using *5 lfinite-LConsI.hyps*(*2*) **by** *blast*
have *7*: *lfinite* (*lfuse xs* (*lfusecat xss*))
by (*simp add*: *1 6 lfuse-lfinite*)
show *?thesis* **by** (*simp add*: *7*)
qed
qed
qed

lemma *lfusecat-repeat-LNil* [*simp*]:
lfusecat (*repeat LNil*) = *LNil*
by (*simp add*: *lfusecat-eq-LNil*)

lemma *llastfirst-lfusecat-llast*:
assumes *lfinite xss*

```
        ¬lnull xss
        ∀ xs ∈ lset xss. ¬lnull xs
        llastlfirst xss
        ∀ xs ∈ lset xss. lfinite xs
 shows    llast(lfusecat xss) = llast(llast xss)
using assms
proof (induction xss)
case lfinite-LNil
then show ?case
using llist.disc(1) by blast
next
case (lfinite-LConsI xss xs)
then show ?case
  proof (cases xss = LNil )
  case True
  then show ?thesis
  by simp
  next
  case False
  then show ?thesis
    proof −
     have 1: llastlfirst xss
       using False lfinite-LConsI.prems(3) llastlfirst-LCons llist.collapse(1) by blast
     have 2: ∀ xs ∈ lset xss. lfinite xs
       by (simp add: lfinite-LConsI.prems(4))
     have 3: ∀ xs ∈ lset xss. ¬lnull xs
        using lfinite-LConsI.prems(2) by auto
     have 4: lfinite xss
       by (simp add: lfinite-LConsI.hyps)
     have 5: llast (lfusecat xss) = llast (llast xss)
       using 1 2 3 False lfinite-LConsI.IH llist.collapse(1) by blast
     have 6: llast (llast (LCons xs xss)) = llast (llast xss)
        by (metis False llast-LCons llist.collapse(1))
     have 7: (lfusecat (LCons xs xss)) = lfuse xs (lfusecat xss)
       by simp
     have 8: ¬ lnull xs
        by (simp add: lfinite-LConsI.prems(2))
     have 9: lfinite xs
        by (simp add: lfinite-LConsI.prems(4))
     have 10: ¬ lnull (lfusecat xss)
        using 3 False lfusecat-not-lnull-var llist.collapse(1) by blast
     have 11: lfinite (lfusecat xss)
        using 2 4 lfusecat-lfinite-a by blast
     have 111: lfirst (lfusecat xss) = lfirst (lfirst xss)
       by (metis 3 False lfirst-def lfirst-lfusecat-lfirst llist.collapse(1) llist.set-sel(1))
     have 12: llast xs = lfirst (lfusecat xss)
       using 10 111 lfinite-LConsI.prems(3) lfusecat-not-lnull llastlfirst-LCons by auto
     have 13: llast (lfuse xs (lfusecat xss)) = llast (lfusecat xss)
        using llast-lfuse[of xs (lfusecat xss) ] 8 9 10 11 12 by blast
     have 14: llast (lfusecat (LCons xs xss)) = llast (lfusecat xss)
```

```
        by (simp add: 13)
     show ?thesis
     using 13 5 6 by auto
   qed
  qed
qed


lemma lfusecat-llength-lNil:
 llength (lfusecat LNil) = 0
by simp

lemma lfusecat-llength-lfinite:
 assumes lfinite xss
         ¬lnull xss
         ∀ xs ∈ lset xss. ¬ lnull xs
         ∀ xs ∈ lset xss. lfinite xs
         llastlfirst xss
 shows   llength(lfusecat xss) =
          eSuc(∑ i = 0 .. (the-enat(epred(llength xss))) . epred(llength (lnth xss i)))
using assms
proof (induction xss)
case lfinite-LNil
then show ?case by simp
next
case (lfinite-LConsI xss xs)
then show ?case
  proof (cases xss = LNil)
  case True
  then show ?thesis
    proof −
     have 1: lfusecat (LCons xs xss) = xs
       by (simp add: True)
     have 2: llength (lfusecat (LCons xs xss) ) = llength xs
       using 1 by auto
     have 4: (∑ i = 0 .. (the-enat(epred(llength(LCons xs xss))))).
                  epred (llength (lnth (LCons xs xss) i))) =
                      epred(llength (lnth (LCons xs xss) 0))
       using True  by simp
     have 5: eSuc(epred(llength (lnth (LCons xs xss) 0))) = llength xs
       by (simp add: lfinite-LConsI.prems(2))
     show ?thesis
     using 2 4 5 by presburger
    qed
  next
  case False
  then show ?thesis
   proof −
   have 6: (lfusecat (LCons xs xss)) = lfuse xs (lfusecat xss)
     by simp
```

```
  have 7: llastlfirst (LCons xs xss)
    by (simp add: lfinite-LConsI.prems(4))
  have 71: lfirst (lfusecat xss) = lfirst(lfirst xss)
    by (metis False insert-iff lfinite-LConsI.prems(2) lfirst-def lfirst-lfusecat-lfirst
        llist.collapse(1) llist.set-sel(1) llist.simps(19))
  have 8: llast xs = lfirst (lfusecat xss)
    using 7 71 False llastlfirst-LCons llist.collapse(1) by auto
  have 9: llength (lfuse xs (lfusecat xss)) = llength xs + epred(llength(lfusecat xss))
    by (simp add: lfinite-LConsI.prems(2) lfuse-llength)
  have 10: (llength(lfusecat xss)) =
           eSuc(∑ i = 0 .. (the-enat(epred (llength xss))). epred(llength (lnth xss i)))
     using 7 False lfinite-LConsI.IH lfinite-LConsI.prems(2) lfinite-LConsI.prems(3) llastlfirst-LCons
     llist.collapse(1) by auto
  have 11: (the-enat(epred(llength (LCons xs xss)))) = (1+the-enat(epred(llength xss)))
     using False
     by simp
       (metis False co.enat.sel(2) eSuc-enat lfinite.cases lfinite-LConsI.hyps lfinite-llength-enat
        llength-LCons  the-enat.simps)
  have 12: (∑ i = 0 .. (the-enat(epred(llength (LCons xs xss)))). epred (llength (lnth (LCons xs xss) i)))
=
           epred(llength(lnth (LCons xs xss) 0)) +
           (∑ i = 1 ..(1+the-enat(epred(llength xss))). epred (llength (lnth (LCons xs xss) i)))
     using 11
     by (simp add: sum.atLeast-Suc-atMost)
  have 13: epred(llength(lnth (LCons xs xss) 0)) = epred (llength xs)
    by simp
  have 14: (∑ i = 1 ..(1+the-enat(epred(llength xss))). epred (llength (lnth (LCons xs xss) i))) =
           (∑ i = 0 ..(the-enat(epred(llength xss))). epred (llength (lnth (LCons xs xss) (Suc i))))
     using sum.shift-bounds-cl-nat-ivl[of λk. epred (llength (lnth (LCons xs xss) k)) 0 1
           (the-enat(epred(llength xss)))]
      by simp
  have 15: (∑ i = 0 ..(the-enat(epred(llength xss))). epred (llength (lnth (LCons xs xss) (Suc i)))) =
           (∑ i = 0 ..(the-enat(epred(llength xss))). epred (llength (lnth (xss) (i))))
     by auto
  have 16: epred (llength xs) + (∑ i = 0 ..(the-enat(epred(llength xss))). epred (llength (lnth (xss) (i))))
=
           (∑ i = 0 .. (the-enat(epred(llength (LCons xs xss)))). epred (llength (lnth (LCons xs xss) i)))
     using 12 13 14 15 by presburger
  have 17: llength xs + epred(llength(lfusecat xss)) =
             eSuc(epred (llength xs) + (∑ i = 0 ..(the-enat(epred(llength xss))). epred (llength (lnth (xss)
(i)))))
       by (metis 10 eSuc-epred eSuc-plus epred-eSuc lfinite-LConsI.prems(2) llength-eq-0 lset-intros(1))
  show ?thesis
  using 16 17 6 9 by presburger
 qed
qed

lemma llastlfirst-ltake:
 assumes n≤llength xss
```

```
        llastlfirst xss
 shows  llastlfirst (ltake (enat n) xss)
using assms
proof (induction n arbitrary: xss)
case 0
then show ?case
by (simp add: llastlfirst-def)
next
case (Suc n)
then show ?case
 unfolding llastlfirst-def
  by auto
   (metis Suc-ile-eq enat-ord-simps(2) less-Suc-eq lnth-ltake order-le-less-trans)
qed

lemma lfusecat-ltake:
 assumes¬ lnull xss
        n ≤ llength xss
       ∀ xs ∈ lset xss. ¬ lnull xs
       ∀ xs ∈ lset xss. lfinite xs
       llastlfirst xss
  shows lfusecat (ltake (enat n) xss) =
        ltake (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
             (lfusecat xss)
proof −
 have 1: lfinite (ltake (enat n) xss)
    using enat-ord-code(4) lfinite-ltake by blast
 have 2: llength (ltake (enat n) xss) = n
   by (simp add: assms(2))
 have 3: (the-enat(epred(llength (ltake (enat n) xss)))) = n−1
    using 2 epred-enat the-enat.simps by presburger
 have 4: ∀ xs ∈ lset (ltake (enat n) xss). ¬ lnull xs
    by (meson assms(3) lset-ltake subsetD)
 have 5: ∀ xs ∈ lset (ltake (enat n) xss). lfinite xs
    by (meson assms(4) lset-ltake subsetD)
 have 6: llastlfirst (ltake (enat n) xss)
   by (simp add: assms(2) assms(5) llastlfirst-ltake)
 have 7: ⋀ j. 0< n ∧ j < n−1 ⟶
             epred(llength (lnth (ltake (enat n) xss) j)) =
             epred(llength (lnth xss j))
     by (metis diff-le-self dual-order.strict-trans1 enat-ord-simps(2) lnth-ltake)
 have 71: llength (lfusecat (ltake (enat n) xss)) =
        (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth (ltake (enat n) xss) i))))
      using lfusecat-llength-lfinite[of (ltake (enat n) xss) ]
      by (metis 1 2 3 4 5 6 lfusecat-LNil llength-LNil llist.collapse(1) ltake-0
      the-enat.simps zero-enat-def)
 have 8: (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth (ltake (enat n) xss) i)))) =
        (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
  using 7 atLeastAtMost-iff[of - 0 n−1] sum.cong[of {0..n−1}
   {0..n−1} λi. epred(llength (lnth (ltake (enat n) xss) i))
```

```
        λi. epred(llength (lnth xss i)) ]
  by (simp add: Suc-ile-eq dual-order.refl lnth-ltake)
 have 9: llength (lfusecat (ltake (enat n) xss)) ≤ llength (lfusecat xss)
    by (metis  lfuse-llength-le-a lfusecat-split)
 have 10: llength (ltake (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
             (lfusecat xss)) =
          (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
   using 71 8 9 by auto
 have 11: ltake (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
              (lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))) =
             (lfusecat (ltake n xss))
    using 71 8 ltake-lfuse[of (lfusecat (ltake (enat n) xss)) (lfusecat (ldrop n xss))]
      by presburger
 have 12: (lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))) = lfusecat xss
   by (metis  lfusecat-split)
 show ?thesis
 using 11 12 by auto
qed

lemma lfusecat-ldrop:
 assumes¬ lnull xss
        n < llength xss
       ∀ xs ∈ lset xss. ¬ lnull xs
       ∀ xs ∈ lset xss. lfinite xs
       llastlfirst xss
  shows  lfusecat (ldrop (enat n) xss) =
         ldrop (if n = 0 then 0 else (∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
             (lfusecat xss)
proof −
 have 1: lfinite (ltake (enat n) xss)
    using enat-ord-code(4) lfinite-ltake by blast
 have 2: llength (ltake (enat n) xss) = n
   by (simp add: assms(2))
 have 3: (the-enat(epred(llength (ltake (enat n) xss)))) = n−1
    using 2 epred-enat the-enat.simps by presburger
 have 4: ∀ xs ∈ lset (ltake (enat n) xss). ¬ lnull xs
    by (meson assms(3) lset-ltake subsetD)
 have 5: ∀ xs ∈ lset (ltake (enat n) xss). lfinite xs
    by (meson assms(4) lset-ltake subsetD)
 have 6: llastlfirst (ltake (enat n) xss)
   by (simp add: assms(2) assms(5) llastlfirst-ltake order.strict-implies-order)
 have 7:  ⋀ j. 0< n ∧ j < n−1 ⟶
             epred(llength (lnth (ltake (enat n) xss) j)) =
             epred(llength (lnth xss j))
    by (metis diff-le-self dual-order.strict-trans1 enat-ord-simps(2) lnth-ltake)
 have 71: llength (lfusecat (ltake (enat n) xss)) =
         (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth (ltake (enat n) xss) i))))
      using lfusecat-llength-lfinite[of (ltake (enat n) xss) ]
      by (metis 1 2 3 4 5 6 lfusecat-LNil llength-LNil llist.collapse(1) ltake-0
          the-enat.simps zero-enat-def)
```

have *8*: (*if n =0 then 0 else eSuc*($\sum$ *i = 0 .. (n−1) . epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*)))) =
(*if n =0 then 0 else eSuc*($\sum$ *i = 0 .. (n−1) . epred*(*llength* (*lnth xss i*))))
using *7 atLeastAtMost-iff*[*of - 0 n−1*] *sum.cong*[*of* {*0..n−1*} {*0..n−1*}
λi. *epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*))
λi. *epred*(*llength* (*lnth xss i*))]
by (*simp add*: *Suc-ile-eq dual-order.refl lnth-ltake*)
have *9*: *llength* (*lfusecat* (*ltake* (*enat n*) *xss*)) ≤ *llength* (*lfusecat xss*)
by (*metis lfuse-llength-le-a lfusecat-split*)
have *10*: *llength* (*ltake* (*if n =0 then 0 else eSuc*($\sum$ *i = 0 .. (n−1) . epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) =
(*if n =0 then 0 else eSuc*($\sum$ *i = 0 .. (n−1) . epred*(*llength* (*lnth xss i*))))
using *7 1 8 9* **by** *auto*
have *11*: *ltake* (*if n =0 then 0 else eSuc*($\sum$ *i = 0 .. (n−1) . epred*(*llength* (*lnth xss i*))))
(*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) =
(*lfusecat* (*ltake n xss*))
using *7 1 8 ltake-lfuse*[*of* (*lfusecat* (*ltake* (*enat n*) *xss*)) (*lfusecat* (*ldrop n xss*))]
by *presburger*
have *12*: *ldrop 0* (*lfuse* (*lfusecat* (*ltake 0 xss*)) (*lfusecat* (*ldrop 0 xss*))) =
(*lfusecat* (*ldrop 0 xss*))
by *simp*
have *13*: *0*< *n* ⟹¬ *lnull* (*lfusecat* (*ltake* (*enat* (*n*::*nat*)) (*xss*::*'a llist llist*)))
using *7 1 8* **by** *force*
have *14*: *0*< *n* ⟹ ¬ *lnull* (*lfusecat* (*ldrop* (*enat n*) *xss*))
by (*meson assms*(*2*) *assms*(*3*) *in-lset-ldropD leD lfusecat-not-lnull-var lnull-ldrop*)
have *15*: *0*<*n* ⟹ *llast* (*lfusecat* (*ltake* (*enat n*) *xss*)) = *lfirst* (*lfusecat* (*ldrop* (*enat n*) *xss*))
proof −
assume *a*: *0*<*n*
have *151*: *llast* (*lfusecat* (*ltake* (*enat n*) *xss*)) = *llast*(*llast* (*ltake* (*enat n*) *xss*))
using *1 13 4 5 6 a lfusecat-not-lnull llastfirst-lfusecat-llast* **by** *blast*
have *152*: *lfirst* (*lfusecat* (*ldrop* (*enat n*) *xss*)) = *lfirst* (*lfirst* (*ldrop* (*enat n*) *xss*))
by (*metis assms*(*2*) *assms*(*3*) *in-lset-conv-lnth ldrop-enat leD lfirst-def lfirst-lfusecat-lfirst*
lhd-ldropn lnull-ldrop)
have *153*: *llast*(*llast* (*ltake* (*enat n*) *xss*)) = *lfirst* (*lfirst* (*ldrop* (*enat n*) *xss*))
using *assms a*
by (*metis 1 13 lappend-ltake-ldrop leD lfusecat-not-lnull llastlfirst-lappend-lfinite*
lnull-ldrop)
show *?thesis*
by (*simp add*: *151 152 153*)
qed
have *16*: *0*<*n* ⟹ *ldrop* (*epred* (*eSuc*($\sum$ *i = 0 .. (n−1) . epred*(*llength* (*lnth xss i*)))))
(*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) =
(*lfusecat* (*ldrop n xss*))
using *7 1 8 13 14 15 ldrop-lfuse-a*[*of* (*lfusecat* (*ltake* (*enat n*) *xss*)) (*lfusecat* (*ldrop n xss*))]
by (*metis 1 5 less-numeral-extra*(*3*) *lfusecat-lfinite-a*)
have *17*: *ldrop* (*if n =0 then 0 else* ($\sum$ *i = 0 .. (n−1) . epred*(*llength* (*lnth xss i*))))
(*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) =
(*lfusecat* (*ldrop n xss*))
using *11 16* **by** *auto*
show *?thesis*
by (*metis 17 lfusecat-split*)

qed

lemma *lfusecat-eq-LCons-conv*:
shows *lfusecat xss = LCons x xs* ⟷
(∃ *xs′ xss′ xss″*. *xss = lappend* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*) ∧
xs = lappend xs′ (*ltl* (*lfusecat xss″*)) ∧ *set xss′* ⊆ {*xs*. *lnull xs*})
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *?rhs*
then obtain *xs′ xss′ xss″*
where *xss = lappend* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*)
and *xs* = ((*lappend xs′* (*ltl* (*lfusecat xss″*))))
and *set xss′* ⊆ {*xs*. *lnull xs*} **by** *blast*
moreover from ‹*set xss′* ⊆ {*xs*. *lnull xs*}›
have *lnull* (*lfusecat* (*llist-of xss′*))
by (*metis lfusecat-not-lnull lset-llist-of mem-Collect-eq subset-eq*)
have *1*: *lfusecat xss = lfuse* (*lfusecat* (*llist-of xss′*)) (*lfusecat* (*LCons* (*LCons x xs′*) *xss″*))
by (*simp add*: *calculation*(*1*) *lfusecat-lappend*)
have *2*: *lfuse* (*lfusecat* (*llist-of xss′*)) (*lfusecat* (*LCons* (*LCons x xs′*) *xss″*)) =
(*lfusecat* (*LCons* (*LCons x xs′*) *xss″*))
by (*simp add*: ‹*lnull* (*lfusecat* (*llist-of* (*xss′*::*′a llist list*)))› *lfuse-conv-lnull llist.expand*)
have *3*: (*lfusecat* (*LCons* (*LCons x xs′*) *xss″*)) = *lfuse* (*LCons x xs′*) (*lfusecat xss″*)
by *simp*
have *4*: *lfuse* (*LCons x xs′*) (*lfusecat xss″*) =
(*LCons x* (*lappend xs′* (*ltl* (*lfusecat xss″*))))
unfolding *lfuse-def*
using *llist.collapse*(*1*) **by** *force*
ultimately show *?lhs*
using *1 2* **by** *auto*
next
assume *?lhs*
hence ¬ *lnull* (*lfusecat xss*) **by** *simp*
hence ¬ *lset xss* ⊆ {*xs*. *lnull xs*} **using** *lfusecat-eq-LNil lnull-def* **by** *blast*
hence ¬ *lnull* (*lfilter* (λ*xs*. ¬ *lnull xs*) *xss*) **by**(*auto*)
then obtain *y ys* **where** *yss*: *lfilter* (λ*xs*. ¬ *lnull xs*) *xss = LCons y ys*
unfolding *not-lnull-conv* **by** *auto*
from *lfilter-eq-LConsD*[*OF this*]
obtain *us vs* **where** *xss*: *xss = lappend us* (*LCons y vs*)
and *lfinite us*
and *lset us* ⊆ {*xs*. *lnull xs*} ¬ *lnull y*
and *ys*: *ys = lfilter* (λ*xs*. ¬ *lnull xs*) *vs* **by** *blast*
from ‹*lfinite us*› **obtain** *us′* **where** [*simp*]: *us = llist-of us′*
unfolding *lfinite-eq-range-llist-of* **by** *blast*
from ‹*lset us* ⊆ {*xs*. *lnull xs*}› **have** *us*: *lnull* (*lfusecat us*)
using *lfusecat-eq-LNil lnull-def* **by** *blast*
from ‹¬ *lnull y*› **obtain** *y′ ys′* **where** *y*: *y = LCons y′ ys′*
unfolding *not-lnull-conv* **by** *blast*
from ‹*?lhs*› *us* **have** [*simp*]: *y′ = x*

```
      xs = (lappend ys' (ltl (lfusecat vs)))
    unfolding xss y
    by (metis ‹¬ lnull (y::'a llist)› ‹lfinite (us::'a llist llist)› eq-LConsD lfusecat-LCons
        lfusecat-lappend lhd-lfuse y)
       (metis ‹¬ lnull (y::'a llist)› ‹lfusecat (xss::'a llist llist) = LCons (x::'a) (xs::'a llist)›
        eq-LConsD lappend-lnull2 lfusecat-LCons lfusecat-lfilter-neq-LNil ltl-lfuse y ys yss)
  from ‹lset us ⊆ {xs. lnull xs}› ys show ?rhs unfolding xss y by simp blast
qed


lemma not-lnull-expand:
  ¬ lnull xs ⟷ (∃ b. xs = (LCons b LNil)) ∨ (∃ b xs'. xs = (LCons b xs') ∧ ¬ lnull xs')
by (metis lhd-LCons-ltl llist.disc(1) llist.disc(2) llist.expand)


lemma is-LMore-llength:
  (∃ b xs'. xs = (LCons b xs') ∧ ¬ lnull xs') ⟷ 1 < llength xs
by (metis dual-order.strict-implies-not-eq gr-implies-not-zero lhd-LCons-ltl llength-LCons
    llength-eq-0 lstrict-prefix-code(2) lstrict-prefix-code(4) lstrict-prefix-llength-less one-eSuc )


lemma is-LEmpty-llength:
  (∃ b. xs = (LCons b LNil)) ⟷ llength xs =1
by (metis epred-llength llength-LCons llength-eq-0 llist.disc(1) ltl-simps(2) not-lnull-expand
     one-eSuc)

lemma lfusecat-all-llength-one-lfuse:
 assumes ∀ ys ∈ lset (llist-of xss'). llength ys = 1
 shows   lfuse (lfusecat (llist-of xss')) ys =
         lfuse (if ¬ lnull (llist-of xss') then (LCons (lfirst (lfirst (llist-of xss'))) LNil) else LNil) ys

proof −
 have 1: (lfusecat (llist-of xss')) =
         (if ¬ lnull (llist-of xss') then (LCons (lfirst (lfirst (llist-of xss'))) LNil) else LNil)
   using assms
  proof (induction xss')
  case Nil
  then show ?case by simp
  next
  case (Cons as xss')
  then show ?case
    proof (cases xss'=Nil )
    case True
    then show ?thesis using Cons
     by simp (metis is-LEmpty-llength lfirst-def lhd-LCons)
    next
    case False
    then show ?thesis using Cons
       by simp
         (metis is-LEmpty-llength lfirst-def lhd-LCons llength-lnull zero-one-enat-neq(1))
```

```
    qed
  qed
  show ?thesis
  using 1 by presburger
qed

lemma lfusecat-eq-LCons-conv-all-lset-singleton:
 assumes ∀ ys ∈ lset (llist-of xss). llength ys = 1
         ¬ lnull (llist-of xss)
         llastlfirst (llist-of xss)
shows  lset (llist-of xss) = {lfirst (llist-of xss) }
using assms
proof (induction xss)
case Nil
then show ?case by simp
next
case (Cons as xss)
then show ?case
 proof −
  have 1: xss= [] ∨ as ∈ lset (llist-of xss)
    by (metis Cons.prems(1) Cons.prems(3) List.set-insert insert-iff is-LEmpty-llength
        le-numeral-extra(4) lfirst-def lhd-LCons-ltl llast-singleton llastlfirst-LCons llist.set-sel(1)
         llist-of.simps(2) lnull-llist-of lset-llist-of ltl-simps(2) not-in-set-insert not-lnull-llength)
  have 2: xss= [] ⟹ ?thesis
    by (simp add: lfirst-def)
  have 3: as ∈ lset (llist-of xss) ⟹ ?thesis
  by (metis 2 Cons.IH Cons.prems(1) Cons.prems(3) insert-absorb2 insert-iff lfirst-def lhd-LCons
      llastlfirst-LCons llist.simps(19) llist-of.simps(2) lnull-llist-of singletonD)
  show ?thesis
  using 1 2 3 by linarith
 qed
qed


lemma lfusecat-eq-LCons-conv-all-the-same:
 assumes ∀ ys ∈ lset (llist-of xss′). llength ys = 1
         ¬ lnull (llist-of xss′)
         llastlfirst (llist-of xss′)
         (llast (llist-of xss′)) = (LCons x LNil)
          ys ∈ lset (llist-of xss′)
 shows    ys = (LCons x LNil)
proof −
 have 1: lset (llist-of xss′) = { lfirst (llist-of xss′)}
   using assms lfusecat-eq-LCons-conv-all-lset-singleton by blast
 have 2: llast (llist-of xss′) = lnth (llist-of xss′) (the-enat(epred(llength (llist-of xss′))))
    by (metis assms(2) co.enat.exhaust-sel enat-the-enat ile-eSuc infinity-ileE llast-conv-lnth
        llength-eq-0 llength-llist-of)
 have 3: lnth (llist-of xss′) (the-enat(epred(llength (llist-of xss′)))) ∈ lset (llist-of xss′)
   by (metis 2 assms(2) co.enat.exhaust-sel ile-eSuc in-lset-lappend-iff infinity-ileE
       lappend-lbutlast-llast-id llength-eq-0 llength-eq-infty-conv-lfinite llength-lbutlast
```

llength-llist-of lset-intros(1))
have *4*: (*LCons x LNil*) ∈ *lset* (*llist-of xss'*)
using *2 3 assms(4)* **by** *force*
show *?thesis*
using *1 4 assms(5)* **by** *auto*
qed

lemma *lfusecat-eq-LCons-conv-all-the-same-a*:
assumes *lfinite uss*
∀ *ys* ∈ *lset* (*uss*). *llength ys* = *1*
¬ *lnull* (*uss*)
llastlfirst (*uss*)
(*llast* (*uss*)) = (*LCons x LNil*)
ys ∈ *lset* (*uss*)
shows *ys* = (*LCons x LNil*)
using *assms lfusecat-eq-LCons-conv-all-the-same llist-of-list-of*
by (*metis* (*full-types*))

lemma *lfusecat-eq-LCons-conv-alt*:
assumes *llastlfirst xss*
∀ *ys* ∈ *lset xss*. ¬ *lnull ys*
¬ *lnull xs*
shows *lfusecat xss* = *LCons x xs* ⟷
(∃ *xs' xss' xss''*. *xss* = *lappend* (*llist-of xss'*) (*LCons* (*LCons x xs'*) *xss''*) ∧ ¬*lnull xs'* ∧
xs = *lfuse xs'* (*lfusecat xss''*) ∧ *set xss'* ⊆ {*xs*. *lnull* (*ltl xs*)})
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *?rhs*
then obtain *xs' xss' xss''*
where *xss* = *lappend* (*llist-of xss'*) (*LCons* (*LCons x xs'*) *xss''*)
and ¬*lnull xs'*
and *xs* = (*lfuse xs'* (*lfusecat xss''*))
and *set xss'* ⊆ {*xs*. *lnull* (*ltl xs*)} **by** *blast*
moreover from ‹*set xss'* ⊆ {*xs*. *lnull* (*ltl xs*)}›
have *00*: *llength* (*lfusecat* (*llist-of xss'*)) ≤*1*
by (*metis is-LMore-llength lfusecat-all-empty-or-LNil-a lset-llist-of lset-lltl-llength-var*
ltl-simps(2) mem-Collect-eq not-le-imp-less subset-eq)
have *01*: ∀ *y* ∈ *lset* (*llist-of xss'*). *llength y* = *1*
by (*metis assms(2) calculation(1) calculation(4) dual-order.strict-iff-order is-LMore-llength*
lset-lappend1 lset-llist-of ltl-simps(2) mem-Collect-eq not-lnull-llength subsetD)
have *02*: *lfinite* (*llist-of xss'*)
using *lfinite-llist-of* **by** *blast*
have *03*: *llastlfirst* (*lappend* (*llist-of xss'*) (*LCons* (*LCons x xs'*) *xss''*))
using *assms calculation(1)* **by** *blast*
have *04*: *llastlfirst* (*llist-of xss'*)
using *assms(1) calculation(1) lfinite-llist-of llastlfirst-lappend-lfinite* **by** *blast*
have *05*: (*if lnull* (*llist-of xss'*) ∨ *lnull* (*LCons* (*LCons x xs'*) *xss''*)
then True

```
        else llast (llast (llist-of xss')) = lfirst (lfirst (LCons (LCons x xs') xss''))
    using 03 02 llastlfirst-lappend-lfinite[of (llist-of xss') (LCons (LCons x xs') xss'') ]
    by blast
  have 06: ¬ lnull (LCons (LCons x xs') xss'')
    by simp
  have 07: ¬ lnull (llist-of xss') ⟹
        llast (llast (llist-of xss')) = lfirst (lfirst (LCons (LCons x xs') xss''))
   using 05 06 by presburger
  have 08: ¬ lnull (llist-of xss') ⟹ (llast (llist-of xss')) = (LCons x LNil)
   using 07 lappend-lbutlast-llast-id[of (llist-of xss') ]
   by (metis 01 02 eq-LConsD in-lset-lappend-iff is-LEmpty-llength lbutlast-lfinite lfirst-def
       llast-singleton lset-intros(1))
  have 09: ¬ lnull (llist-of xss') ⟹ (∀ ys ∈ lset (llist-of xss'). ys = (LCons x LNil))
    using lfusecat-eq-LCons-conv-all-the-same
    by (metis 01 04 08)
  have 0: (lfusecat (llist-of xss')) = LNil ∨ (lfusecat (llist-of xss')) = (LCons x LNil)
    by (metis 00 09 eq-LConsD is-LMore-llength lfirst-def lfirst-lfusecat lfusecat-LNil lhd-LCons-ltl
       linorder-not-le llist.collapse(1) llist.set-sel(1))
  have 1: lfusecat xss = lfuse (lfusecat (llist-of xss')) (lfusecat (LCons (LCons x xs') xss''))
   by (simp add: calculation(1) lfusecat-lappend)
  have 2: lfuse (lfusecat (llist-of xss')) (lfusecat (LCons (LCons x xs') xss'')) =
          (lfusecat (LCons (LCons x xs') xss''))
       using 0 by fastforce
  have 3: (lfusecat (LCons (LCons x xs') xss'')) = lfuse (LCons x xs') (lfusecat xss'')
   by simp
  have 4: lfuse (LCons x xs') (lfusecat xss'') =
       (LCons x ( (lfuse xs' ( (lfusecat xss'')))
       ))
    unfolding lfuse-def
    using calculation(2) by auto
  ultimately show ?lhs
  using 1 2 by auto
next
  assume ?lhs
  hence ¬ lnull (lfusecat xss) by simp
  hence ¬ lset xss ⊆ {xs. lnull (ltl xs)}
  by (metis ‹lfusecat (xss::'a llist llist) = LCons (x::'a) (xs::'a llist)› assms(3)
      lfusecat-all-empty-or-LNil-a lnull-ldrop lset-lltl-llength-var ltl-ldrop-one ltl-simps(2)
      mem-Collect-eq subsetD)
  hence ¬ lnull (lfilter (λxs. ¬ lnull (ltl xs)) xss) by(auto)
  then obtain y ys where yss: lfilter (λxs. ¬ lnull (ltl xs)) xss = LCons y ys
    unfolding not-lnull-conv by auto
  from lfilter-eq-LConsD[OF this]
  obtain us vs where xss: xss = lappend us (LCons y vs)
    and lfinite us
    and lset us ⊆ {xs. lnull (ltl xs)} ¬ lnull (ltl y)
    and ys: ys = lfilter (λxs. ¬ lnull (ltl xs)) vs  using lnull-ltlI by blast
  from ‹lfinite us› obtain us' where [simp]: us = llist-of us'
    unfolding lfinite-eq-range-llist-of by blast
  from ‹lset us ⊆ {xs. lnull (ltl xs)}› have us: llength (lfusecat us) ≤1
```

using *lfusecat-eq-LNil*
by (*metis lfusecat-all-empty-or-LNil-a lnull-ldrop lset-lltl-llength-var ltl-ldrop-one mem-Collect-eq subset-eq*)
from ‹¬ *lnull* (*ltl y*)› **obtain** y' ys' **where** *y*: $y = LCons\ y'\ ys'$
unfolding *not-lnull-conv*
by (*metis* ‹¬ *lnull* (*ltl y*)› *lhd-LCons-ltl lnull-ltlI*)
have *100*: *llastlfirst us*
using ‹*lfinite us*› *assms*(*1*) *llastlfirst-lappend-lfinite xss* **by** *blast*
have *101*: ¬ *lnull* ys'
using ‹¬ *lnull* (*ltl y*)› *y* **by** *auto*
have *102*: ¬ *lnull* (*LCons y vs*)
by *simp*
have *103*: ¬ *lnull us* ⟹ *llast* (*llast us*) = *lfirst* (*lfirst* (*LCons y vs*))
using *llastlfirst-lappend-lfinite*[*of us* (*LCons y vs*)]
using *assms*(*1*) *xss* **by** *auto*
have *104*: (∀ *z* ∈ *lset us*. *llength z* =*1*)
by (*metis* ‹*lset us* ⊆ {*xs*. *lnull* (*ltl xs*)}› *assms*(*2*)
dual-order.strict-iff-order eq-LConsD in-lset-lappend-iff is-LMore-llength mem-Collect-eq not-lnull-llength subsetD xss)
have *1040*: ¬ *lnull us* ⟹ *llast us* ∈ *lset us*
by *simp*
have *1041*: ¬ *lnull us* ⟹ *llast us* = *LCons x LNil*
using *lappend-lbutlast-llast-id*[*of us*] *100 104 1040*
lfusecat-eq-LCons-conv-all-the-same-a[*of us x llast us*]
lfusecat-eq-LCons-conv-all-lset-singleton[*of us'*]
by (*metis* ‹¬ *lnull* (*lfusecat xss*)› ‹*lfusecat xss* = *LCons x xs*› ‹*us* = *llist-of us'*›
eq-LConsD is-LEmpty-llength lfirst-def lfirst-lfusecat-lfirst lfusecat-not-lnull lhd-lappend lset-eq-empty not-lnull-lset-conv-a singletonD xss)
have *105*: ¬ *lnull us* ⟹ (∀ *z* ∈ *lset us*. *z*= (*LCons x LNil*))
using *lfusecat-eq-LCons-conv-all-the-same-a*[*of us x*]
using *100 104 1041* ‹*lfinite us*› **by** *blast*
have *106*: ¬ *lnull us* ⟹ *lfusecat us* = (*LCons x LNil*)
by (*metis 100 104 1041* ‹*lfinite us*› *dual-order.antisym is-LEmpty-llength lfinite-LConsI lfinite-LNil lfusecat-not-lnull-var llast-singleton llastfirst-lfusecat-llast llength-LNil lset-eq-empty not-lnull-llength not-lnull-lset-conv-a us zero-one-enat-neq*(*1*))
from ‹*?lhs*› *us* **have** [*simp*]: $y' = x$
by (*metis 103 1041* ‹¬ *lnull* (*ltl y*)› *eq-LConsD lappend-lnull1 lfirst-def lfirst-lfusecat llast-singleton lnull-ltlI xss y*)
from ‹*?lhs*› *us* **have** [*simp*]: *xs* = ((*lfuse ys'* ((*lfusecat vs*))))
using *lfusecat-lappend*[*of us* (*LCons y vs*)] *lfusecat-LCons*[*of y vs*] *y xss*
101 ‹*lfinite us*› ‹*lfusecat xss* = *LCons x xs*›
by (*metis 103 1041 106 eq-LConsD lappend-code*(*1*) *lappend-lnull1 lbutlast-simps*(*2*) *lfirst-def lfuse-LCons-a lfuse-conv-lnull lfuse-lbutlast lnull-ltlI*)
from ‹*lset us* ⊆ {*xs*. *lnull* (*ltl xs*)}› *ys* **show** *?rhs* **unfolding** *xss y*
using *101* **by** *auto*
qed

lemma *lfusecat-eq-One-conv*:
lfusecat xss = *LCons x LNil* ⟷

$(\exists\ xss'\ xss''.\ xss = lappend\ (llist\text{-}of\ xss')\ (LCons\ (LCons\ x\ LNil)\ xss'') \wedge$
$LNil = (ltl\ (lfusecat\ xss'')) \wedge set\ xss' \subseteq \{xs.\ lnull\ xs\})$
by (*metis lappend-eq-LNil-iff lfusecat-eq-LCons-conv*)

lemma *llength-lfusecat-eq-one-conv*:
llength (*lfusecat xss*) = *1* $\longleftrightarrow$
$(\exists\ x\ xss'\ xss''.\ xss = lappend\ (llist\text{-}of\ xss')\ (LCons\ (LCons\ x\ LNil)\ xss'') \wedge$
$LNil = (ltl\ (lfusecat\ xss'')) \wedge set\ xss' \subseteq \{xs.\ lnull\ xs\})$
by (*metis is-LEmpty-llength lfusecat-eq-One-conv*)

lemma *lfusecat-lfinite-b*:
assumes *lfinite*(*lfusecat xss*)
$\forall\, xs \in lset\ xss.\ \neg\ lnull\ (ltl\ xs)$
$\forall\, xs \in lset\ xss.\ lfinite\ xs$
llastlfirst xss
shows *lfinite xss*
using *assms*
proof (*induct zs*≡(*lfusecat xss*) *arbitrary*: *xss*)
case *lfinite-LNil*
then show *?case*
by (*metis lfusecat-not-lnull-var llist.disc*(*1*) *lnull-imp-lfinite lset-eq-empty ltl-simps*(*1*)
not-lnull-lset-conv-a)
next
case (*lfinite-LConsI xs x*)
then show *?case*
proof −
have *1*: $(\exists\, xs'\ xss'\ xss''.\ xss = lappend\ (llist\text{-}of\ xss')\ (LCons\ (LCons\ x\ xs')\ xss'') \wedge$
$xs = lappend\ xs'\ (ltl\ (lfusecat\ xss'')) \wedge set\ xss' \subseteq \{xs.\ lnull\ xs\})$
using *lfinite-LConsI.hyps*(*3*) *lfusecat-eq-LCons-conv* **by** *fastforce*
obtain *xs' xss' xss''* **where** *2*: $xss = lappend\ (llist\text{-}of\ xss')\ (LCons\ (LCons\ x\ xs')\ xss'') \wedge$
$xs = lappend\ xs'\ (ltl\ (lfusecat\ xss'')) \wedge set\ xss' \subseteq \{xs.\ lnull\ xs\}$
using *1* **by** *blast*
have *3*: (*llist-of xss'*) = *LNil*
by (*metis 2 in-lset-lappend-iff lconcat-eq-LNil lfinite-LConsI.prems*(*1*) *llist.collapse*(*1*)
llist.set-sel(*1*) *lnull-lconcat lset-eq-forall-lnull lset-llist-of ltl-simps*(*1*))
have *4*: *xss* = (*LCons* (*LCons x xs'*) *xss''*)
by (*simp add*: *2 3*)
have *5*: *llastlfirst xss*
by (*simp add*: *lfinite-LConsI.prems*(*3*))
have *6*: ¬ *lnull xs'*
using *4 lfinite-LConsI.prems*(*1*) **by** *force*
have *7*: *lnull xss''* $\Longrightarrow$ *?thesis*
by (*simp add*: *4*)
have *8*: ¬ *lnull xss''* $\Longrightarrow$ *?thesis*
proof −
assume *a*: ¬*lnull xss''*
have *9*: *llast* (*LCons x xs'*) = *lfirst* (*lfirst* (*xss''*))
using *a 6 4 5 llastlfirst-LCons* **by** *blast*
have *10*: $\forall\, xs \in lset\ xss''.\ \neg\ lnull\ (ltl\ xs)$

```
      by (simp add: 4 lfinite-LConsI.prems)
    have 11: ∀ xs ∈ lset xss''. lfinite xs
      by (simp add: 4 lfinite-LConsI.prems)
    have 12: llastlfirst xss''
      using 4 5 a by auto
    have 13: lfinite(lfusecat xss'')
      using 2 lfinite-LConsI.hyps(1) by auto
    have 14: xs = lappend (lbutlast xs') (lfusecat xss'')
      by (metis 10 2 6 9 a lappend-lbutlast-llast-id lappend-snocL1-conv-LCons2 lfirst-def
          lfirst-lfusecat-lfirst lfusecat-not-lnull-var lhd-LCons-ltl llast-LCons llist.disc(1)
          llist.set-sel(1) lset-eq-empty ltl-simps(1) not-lnull-lset-conv-a)
    have 15: xs= lfuse xs' (lfusecat xss'')
      by (simp add: 2 6 lappend-lnull2 lfuse-def)
    have 16:  xs = lfusecat (LCons xs' xss'')
      by (simp add: 15)
    have 17: lnull (ltl xs') ⟹ ?thesis
      by (metis 10 11 12 14 4 lappend-code(1) lbutlast.ctr(1) lfinite.lfinite-LConsI
          lfinite-LConsI.hyps(2) llist.collapse(1))
    have 18: ¬lnull (ltl xs') ⟹ (∀ xs ∈ lset (LCons xs' xss''). ¬ lnull (ltl xs))
      by (simp add: 10)
    have 19: ∀ xs ∈ lset (LCons xs' xss''). lfinite xs
      by (metis 11 2 insert-iff lappend-inf lfinite-LConsI.hyps(1) llist.simps(19))
    have 20: llastlfirst (LCons xs' xss'')
      by (metis 12 6 9 a llast-LCons llastlfirst-LCons)
    have 21: lfinite (LCons xs' xss'')
      by (metis 16 17 18 19 20 4 lfinite-LConsI.hyps(2) lfinite-code(2) )
    show ?thesis
    using 21 4 by auto
  qed
 show ?thesis
 using 7 8 by blast
 qed
qed

lemma lfusecat-lfinite:
assumes ∀ xs ∈ lset xss. ¬ lnull (ltl xs)
        ∀ xs ∈ lset xss. lfinite xs
        llastlfirst xss
shows lfinite (lfusecat xss) ⟷ lfinite xss
using assms lfusecat-lfinite-a lfusecat-lfinite-b by blast

lemma ridx-lfusecat-ltake:
 assumes ridx R (lfusecat xss)
         n ≤ llength xss
 shows    ridx R (lfusecat (ltake n xss))
using assms
by (metis lfusecat-split ltake-all ltake-lfuse nle-le ridx-ltake-a)


lemma ridx-lfusecat-ldrop:
```

assumes *ridx R* (*lfusecat xss*)
(*enat n*) < *llength xss*
llastlfirst xss
∀ *xs* ∈ *lset xss*. ¬ *lnull xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
shows *ridx R* (*lfusecat* (*ldrop n xss*))
proof −
have *1*: (*lfusecat* (*ldrop n xss*)) =
ldrop (*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)
using *assms gr-implies-not-zero lfusecat-ldrop llength-eq-0* **by** *blast*
have *2*: (*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) < *llength* (*lfusecat xss*)
by (*metis* (*no-types*, *lifting*) *1 add.commute add.right-neutral assms*(*2*) *assms*(*4*) *in-lset-conv-lnth*
lfusecat-not-lnull llist.set-sel(*1*) *lnth-0-conv-lhd lnth-ldrop lnull-ldrop not-less-iff-gr-or-eq*
order.order-iff-strict the-enat.simps zero-enat-def)
have *3*: ∃ *k*. (*enat k*) = (*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
proof (*cases n*)
case *0*
then show *?thesis* **by** (*simp add*: *zero-enat-def*)
next
case (*Suc nat*)
then show *?thesis* **by** (*metis* (*mono-tags*, *lifting*) *2 enat-iless enat-less-imp-le leD*)
qed
have *4*: *ridx R* (*ldrop* (*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*))
using *ridx-ldrop*[*of R* (*lfusecat xss*)]
using *2 assms*(*1*) *order.strict-implies-order* **by** *blast*
show *?thesis* **by** (*simp add*: *1 4*)
qed

lemma *ridx-lfusecat-a*:
assumes *llastlfirst xss*
∀ *xs* ∈ *lset xss*. ¬ *lnull xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
ridx R (*lfusecat xss*)
i < *llength xss*
shows *ridx R* (*lnth xss i*)
proof −
have *1*: *lsub i i xss* = (*LCons* (*lnth xss i*) *LNil*)
by (*simp add*: *assms*(*5*) *lsub-same*)
have *2*: *lsub i i xss* = *ltake* (*eSuc* (*enat* (*i* − *i*))) (*ldrop* (*enat i*) *xss*)
by (*simp add*: *lsub-def*)
have *3*: *ltake* (*eSuc* (*enat* (*i* − *i*))) (*ldrop* (*enat i*) *xss*) = *ltake 1* (*ldrop* (*enat i*) *xss*)
by (*simp add*: *eSuc-enat one-enat-def*)
have *4*: *ridx R* (*lfusecat* (*ltake 1* (*ldrop* (*enat i*) *xss*)))
by (*metis assms ltake-all nle-le ridx-lfusecat-ldrop ridx-lfusecat-ltake*)
have *5*: (*lfusecat* (*ltake 1* (*ldrop* (*enat i*) *xss*))) = (*lnth xss i*)
by (*metis 1 2 3 lfuse-LNil-2 lfusecat-LCons lfusecat-LNil*)
show *?thesis*
using *4 5* **by** *auto*

qed

lemma *ridx-lfuse-lfinite*:
assumes *lfinite l1*
llast l1 = lfirst l2
shows *ridx R (lfuse l1 l2) ⟷ ridx R l1 ∧ ridx R l2*
using *assms*
proof (*cases ¬ lnull l1 ∧ ¬ lnull l2*)
case *True*
then show *?thesis*
proof (*cases ¬ lnull (ltl l2)*)
case *True*
then show *?thesis*
proof −
have *1*: (*lfuse l1 l2*) = *lappend l1* (*ltl l2*)
unfolding *lfuse-def* **using** ‹¬ *lnull l1 ∧ ¬ lnull l2*› **by** *simp*
have *2*: *ridx R* (*lappend l1* (*ltl l2*)) = (*ridx R l1 ∧* (*R* (*llast l1*) (*lhd* (*ltl l2*))) *∧ ridx R* (*ltl l2*))
using *assms ridx-lappend-lfinite*[*of l1 R ltl l2*] *True* ‹¬ *lnull l1 ∧ ¬ lnull l2*›
by *auto*
have *3*: (*R* (*llast l1*) (*lhd* (*ltl l2*)) *∧ ridx R* (*ltl l2*)) = *ridx R l2*
using *True* ‹¬ *lnull l1 ∧ ¬ lnull l2*› *ridx-LCons-1*[*of R lfirst l2 ltl l2*]
by (*simp add*: *assms lfirst-def*)
show *?thesis*
by (*simp add*: *1 2 3*)
qed
next
case *False*
then show *?thesis*
proof −
have *4*: (*lfuse l1 l2*) = *lappend l1* (*ltl l2*)
unfolding *lfuse-def* **using** ‹¬ *lnull l1 ∧ ¬ lnull l2*› **by** *simp*
have *5*: *ridx R* (*lappend l1* (*ltl l2*)) = (*ridx R l1 ∧ ridx R l2*)
using *assms ridx-lappend-lfinite*[*of l1 R ltl l2*] *False* ‹¬ *lnull l1 ∧ ¬ lnull l2*›
ridx-expand-1 **by** (*metis lhd-LCons-ltl ridx-LCons-1*)
show *?thesis* **by** (*simp add*: *4 5*)
qed
qed
next
case *False*
then show *?thesis*
by (*metis lfuse-LNil-1 lfuse-LNil-2 llist.collapse*(*1*) *ridx-expand-1*)
qed

lemma *ridx-lfusecat-lfuse*:
assumes *llastlfirst xss*
∀ xs ∈ lset xss. ¬ lnull xs
∀ xs ∈ lset xss. lfinite xs

$\forall i.\ \ i < llength\ xss \longrightarrow\ \ ridx\ R\ (lnth\ xss\ i)$
$(Suc\ n) < llength\ xss$
shows $ridx\ R\ (lfuse\ (lnth\ xss\ n)\ (lnth\ xss\ (Suc\ n)))$
using *assms*
by (*metis Suc-ile-eq lfuse-inf llastlfirst-def order-less-imp-le ridx-lfuse-lfinite*)

lemma *ridx-lfusecat-ltake-a*:
assumes *llastlfirst xss*
$\forall\ xs \in lset\ xss.\ \neg\ lnull\ xs$
$\forall\ xs \in lset\ xss.\ lfinite\ xs$
$\forall i.\ \ i < llength\ xss \longrightarrow\ \ ridx\ R\ (lnth\ xss\ i)$
$n \leq llength\ xss$
shows $ridx\ R\ (lfusecat\ (ltake\ (enat\ n)\ xss))$
using *assms*
proof (*induct n arbitrary*: *xss*)
case *0*
then show *?case*
by (*metis LNil-eq-ltake-iff gr-implies-not-zero lfusecat-LNil llength-eq-0 llist.disc(1) ridx-def zero-enat-def*)
next
case (*Suc n*)
then show *?case*
proof −
have *0*: $n{=}0 \Longrightarrow$ *?thesis*
proof −
assume *a0*: $n{=}0$
have *000*: $\neg lnull\ xss$
using *Suc.prems(5) a0 one-enat-def* **by** *force*
have *001*: $(ltake\ (enat\ (Suc\ n))\ xss) = (LCons\ (lnth\ xss\ 0)\ LNil)$
using *a0 000*
by (*metis One-nat-def lhd-LCons-ltl lnth-0-conv-lhd ltake.ctr(1) ltake-eSuc-LCons one-eSuc one-enat-def*)
have *002*: $lfusecat\ (ltake\ (enat\ (Suc\ n))\ xss) =\ \ (lnth\ xss\ 0)$
using *a0 000 001 lfusecat-LCons*[*of* $(lnth\ xss\ 0)\ LNil$] **by** *simp*
show *?thesis* **using** *002 Suc.prems(4) Suc.prems(5) Suc-ile-eq a0* **by** *auto*
qed
have *01*: $n{>}0 \Longrightarrow$ *?thesis*
proof −
assume *a*: $n{>}0$
have *1*: $(lfusecat\ (ltake\ (enat\ (Suc\ n))\ xss)) =$
$ltake\ (eSuc\ (\sum i{::}nat = 0{::}nat..Suc\ n - (1{::}nat).\ epred\ (llength\ (lnth\ xss\ i))))\ (lfusecat\ xss)$
using *lfusecat-ltake*[*of xss* $(Suc\ n)$]
using *Suc.prems(1) Suc.prems(2) Suc.prems(3) Suc.prems(5) Suc-ile-eq* **by** *force*
have *2*: $ridx\ R\ (lfusecat\ (ltake\ (enat\ n)\ xss))$
using *Suc Suc-ile-eq order.strict-implies-order* **by** *blast*
have *20*: $lfinite\ (ltake\ (enat\ n)\ xss)$
by *simp*
have *21*: $\neg\ lnull\ (ltake\ (enat\ n)\ xss)$
using *Suc.prems(5) a enat-0-iff(2)* **by** *fastforce*

```
    have 22: ∀ xs∈lset (ltake (enat n) xss). ¬ lnull xs
     by (meson Suc.prems(2) lset-ltake subsetD)
    have 23: ∀ xs∈lset (ltake (enat n) xss). lfinite xs
     by (meson Suc.prems(3) lset-ltake subsetD)
    have 24: llastlfirst (ltake (enat n) xss)
     using Suc.prems(1) Suc.prems(5) Suc-ile-eq less-imp-le llastlfirst-ltake by blast
    have 25: llast (lfusecat (ltake (enat n) xss)) = llast (llast (ltake (enat n) xss))
      using 20 21 22 23 24 llastfirst-lfusecat-llast by blast
    have 26: lfinite (llast (ltake (enat n) xss))
       by (metis Suc.prems(3) Suc.prems(5) Suc-ile-eq Suc-pred' a enat-ord-code(4) in-lset-conv-lnth
          lfinite-ltake llast-lappend-LCons llast-singleton ltake-Suc-conv-snoc-lnth order-less-imp-le)
    have 27: n< llength xss
       using Suc.prems(5) Suc-ile-eq order-less-imp-le by blast
    have 271: (llast (ltake (enat n) xss)) =  (lnth xss (n−1))
       by (metis 27 Suc-diff-1 Suc-ile-eq a enat-ord-code(4) lfinite-ltake llast-lappend-LCons
          llast-singleton ltake-Suc-conv-snoc-lnth order.strict-implies-order)
    have 28: llast (llast (ltake (enat n) xss)) = llast (lnth xss (n−1))
      using 271 by auto
    have 29: llast (lnth xss (n−1)) = lfirst (lnth xss n)
       by (metis 27 Suc.prems(1) Suc-pred' a llastlfirst-def)
    have 30: ridx R (lfuse (lfusecat (ltake (enat n) xss)) (lnth xss ( n)))
     by (metis 2 25 27 28 29 Suc.prems(4) lfuse-inf ridx-lfuse-lfinite)
    have 31: (lfuse (lfusecat (ltake (enat n) xss)) (lnth xss ( n))) =
              (lfusecat (ltake (enat (Suc n)) xss))
      by (simp add: 27 lfusecat-lappend ltake-Suc-conv-snoc-lnth)
     show ?thesis
    using 30 31 by auto
  qed
 show ?thesis
  using 0 01 by fastforce
 qed
qed

lemma lfusecat-ltake-llength-less-than-llength-lfusecat:
 assumes llastlfirst xss
       ∀ xs ∈ lset xss. 1 < llength xs
       ∀ xs ∈ lset xss. lfinite xs
        (enat n)≤ llength xss
 shows  min
          (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
           (llength (lfusecat xss)) =
           (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
proof −
 have 0: ∀ xs ∈ lset xss. ¬lnull xs
   using assms(2) by fastforce
 have 1: lfinite xss ⟹ ¬ lnull xss⟹ llength(lfusecat xss) =
        eSuc(∑ i = 0 .. (the-enat(epred(llength xss))) . epred(llength (lnth xss i)))
 using lfusecat-llength-lfinite[of xss] assms 0 by fastforce
 have 2: ¬ lfinite xss ⟹ ¬lfinite (lfusecat xss)
   using assms lfusecat-lfinite[of xss] 0
```

by (*metis less-numeral-extra*(*4*) *lhd-LCons-ltl llength-LCons llength-LNil llist.collapse*(*1*) *one-eSuc*)
have *3*: ¬ *lfinite xss* ⟹ *llength* (*lfusecat xss*) = ∞
using *not-lfinite-llength 2* **by** *blast*
have *50*: ¬ *lnull xss*⟹ *lfusecat* (*ltake* (*enat n*) *xss*) =
ltake (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)
using *assms lfusecat-ltake*[*of xss n*] *0* **by** *fastforce*
have *51*: *lfinite* (*lfusecat* (*ltake* (*enat n*) *xss*))
by (*meson assms*(*3*) *enat-ord-code*(*4*) *lfinite-ltake lfusecat-lfinite-a lset-ltake subsetD*)
have *52*: *lfinite* (*ltake* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*))
using *50 51 llist.collapse*(*1*) **by** *fastforce*
have *53*: ∃ *m*. *llength* (*ltake* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) = (*enat m*)
using *52 lfinite-llength-enat* **by** *blast*
have *54*: *llength* (*ltake* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) ≤ *llength* (*lfusecat xss*)
by *auto*
have *55*: *llength* (*ltake* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) =
min
(*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*))
using *llength-ltake*[*of* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)] **by** *auto*
have *56*: *min*
(*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*)) < ∞
by (*metis 53 55 enat-ord-code*(*4*))
have *57*: (*llength* (*lfusecat xss*)) = *llength* (*lfusecat* (*lappend* (*ltake n xss*) (*ldrop n xss*)))
by (*simp add*: *lappend-ltake-ldrop*)
have *58*: *llength* (*lfusecat* (*lappend* (*ltake n xss*) (*ldrop n xss*))) =
llength (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*)))
by (*metis 57 lfusecat-split*)
have *59*: *lnull* (*lfusecat* (*ltake* (*enat n*) *xss*)) ⟷ *n* = *0*
proof (*cases xss*)
case *LNil*
then show *?thesis* **using** *assms enat-0-iff*(*2*) **by** *force*
next
case (*LCons x21 x22*)
then show *?thesis* **using** *1 2 50* **by** *force*
qed
have *60*: *llength* (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) =
llength (*lfusecat* (*ltake* (*enat n*) *xss*)) +
(*if n*= *0 then llength* (*lfusecat* (*ldrop* (*enat n*) *xss*))
else epred (*llength* (*lfusecat* (*ldrop* (*enat n*) *xss*))))
using *lfuse-llength*[*of* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))]
using *59* **by** *presburger*
have *62*: *n*= *0* ⟹ *llength* (*lfusecat* (*ldrop* (*enat n*) *xss*)) = *llength* (*lfusecat xss*)
using *57 58 59 llist.collapse*(*1*) **by** *force*

have *620*: $n>0 \Longrightarrow n<llength\ xss \Longrightarrow \neg\ lnull\ (ldrop\ (enat\ n)\ xss)$
using *Suc-ile-eq assms* **by** *simp*
have *621*: $n>0 \Longrightarrow n<llength\ xss \Longrightarrow (\exists xs{\in}lset\ (ldrop\ (enat\ n)\ xss).\ \neg\ lnull\ xs)$
by (*meson 0 620 in-lset-ldropD lfusecat-not-lnull lfusecat-not-lnull-var*)
have *622*: $n>0 \Longrightarrow n<llength\ xss \Longrightarrow (\neg\ lnull\ (lfusecat\ (ldrop\ (enat\ n)\ xss)))$
using *620 621 lfusecat-not-lnull*[*of* $(ldrop\ (enat\ n)\ xss)$] **by** *blast*
have *623*: $n>0 \Longrightarrow\ \neg\ lnull\ xss$
using *assms 59* **by** *force*
have *63*: $n>0 \Longrightarrow\ n<llength\ xss \Longrightarrow (llength\ (lfusecat\ (ldrop\ (enat\ n)\ xss))) =$
$(llength\ (ldrop\ ((\sum\ i = 0\ ..\ (n-1)\ .\ epred(llength\ (lnth\ xss\ i))))\ (lfusecat\ xss)\ ))$
using *lfusecat-ldrop*[*of xss n*] *assms*
using *623 llength-LNil not-one-less-zero 0* **by** *presburger*
have *630*: $n>0 \Longrightarrow n<llength\ xss \Longrightarrow epred(llength\ (lfusecat\ (ldrop\ (enat\ n)\ xss))) =$
$epred(llength\ (ldrop\ ((\sum\ i = 0\ ..\ (n-1)\ .\ epred(llength\ (lnth\ xss\ i))))\ (lfusecat\ xss)\ ))$
using *63* **by** *presburger*
have *631*: $n>0 \Longrightarrow n<llength\ xss \Longrightarrow$
$(llength\ (ldrop\ ((\sum\ i = 0\ ..\ (n-1)\ .\ epred(llength\ (lnth\ xss\ i))))\ (lfusecat\ xss)\ )) > 0$
by (*metis 622 63 gr-zeroI llength-eq-0*)
have *64*: $llength\ (lfusecat\ (ltake\ (enat\ n)\ xss)) \leq$
$llength\ (lfuse\ (lfusecat\ (ltake\ n\ xss))\ (lfusecat\ (ldrop\ n\ xss)))$
using *lfuse-llength-le-a* **by** *blast*
have *65*: $0 < n \Longrightarrow n<llength\ xss \Longrightarrow 0< (llength\ (lfusecat\ (ldrop\ (enat\ n)\ xss)))$
using *63 631* **by** *presburger*
have *66*: $n>0 \Longrightarrow llength\ xss > 0$
using *623 gr-zeroI llength-eq-0* **by** *blast*
have *67*: $n>0 \Longrightarrow n-1 \leq\ (epred\ (llength\ xss))$
using *assms 66*
by (*metis epred-enat epred-le-epredI*)
have *68*: *min*
$(if\ n =0\ then\ 0\ else\ eSuc(\sum\ i = 0\ ..\ (n-1)\ .\ epred(llength\ (lnth\ xss\ i))))$
$(llength\ (lfusecat\ xss)) =$
$(if\ n =0\ then\ 0\ else\ eSuc(\sum\ i = 0\ ..\ (n-1)\ .\ epred(llength\ (lnth\ xss\ i))))$
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
proof −
have *681*: $n>0 \Longrightarrow n<llength\ xss \Longrightarrow eSuc(\sum\ i = 0\ ..\ (n-1)\ .\ epred(llength\ (lnth\ xss\ i))) \leq$
$eSuc\ (\sum i = 0..the\text{-}enat\ (epred\ (llength\ xss)).\ epred\ (llength\ (lnth\ xss\ i)))$
by (*metis 1 631 True gr-implies-not-zero ileI1 linorder-not-less llength-lnull lnull-ldrop*)
have *682*: $n>0 \Longrightarrow n=llength\ xss \Longrightarrow eSuc(\sum\ i = 0\ ..\ (n-1)\ .\ epred(llength\ (lnth\ xss\ i))) \leq$
$eSuc\ (\sum i = 0..the\text{-}enat\ (epred\ (llength\ xss)).\ epred\ (llength\ (lnth\ xss\ i)))$
by (*metis dual-order.refl epred-enat the-enat.simps*)
show *?thesis*
using *1 681 682 True 623 assms*(*4*) *min.absorb1 order.order-iff-strict* **by** *auto*
qed
next
case *False*
then show *?thesis* **using** *3 min-enat-simps*(*4*) **by** *presburger*
qed
show *?thesis*

using *68* **by** *blast*
qed

lemma *lfusecat-ltake-llength-less-than-next*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss*. *1* < *llength xs*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
(*Suc n*) < *llength xss*
shows (*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
< (*if* (*Suc n*) =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*) . *epred*(*llength* (*lnth xss i*))))
proof −
have *0*: *n*<∞
by *simp*
have *1*: $\bigwedge$*i*. *i*< *llength xss* $\Longrightarrow$ *1*< *llength* (*lnth xss i*)
by (*meson assms*(*2*) *in-lset-conv-lnth*)
have *01*: $\bigwedge$*i*. *i*< *llength xss* $\Longrightarrow$ ¬*lnull* (*lnth xss i*)
by (*metis 1 llength-LNil llist.collapse*(*1*) *not-one-less-zero*)
have *02*: $\forall$ *xs* $\in$ *lset xss*. ¬*lnull xs*
using *assms*(*2*) **by** *fastforce*
have *2*: $\bigwedge$*i* . *i*< *llength xss* $\Longrightarrow$ *0*<*epred*(*llength* (*lnth xss i*))
by (*metis 1 co.enat.exhaust-sel gr-zeroI less-numeral-extra*(*4*) *not-one-less-zero one-eSuc*)
have *3*: *n*= *0* $\Longrightarrow$ *?thesis*
by *auto*
have *4*: *0*<*n* $\Longrightarrow$ ($\sum$ *i* = *0* .. (*n*) . *epred*(*llength* (*lnth xss i*))) =
($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))) +
epred(*llength* (*lnth xss n*))
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 sum.atLeast0-atMost-Suc*)
have *5*: $\bigwedge$*i*. *i*< *llength xss* $\Longrightarrow$ *epred*(*llength* (*lnth xss i*)) < ∞
using *assms*(*3*)
by (*metis enat-ord-simps*(*4*) *epred-0 epred-Infty epred-inject in-lset-conv-lnth infinity-ne-i0*
llength-eq-infty-conv-lfinite)
have *50*: *llength* (*lfusecat* (*ltake n xss*)) $\leq$ *llength*(*lfusecat xss*)
by (*simp add*: *lprefix-lfusecatI lprefix-llength-le*)

have *6*: *0*< *n* $\Longrightarrow$ *eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))< ∞
proof −
assume *a*: *n*>*0*
have *60*: (*lfusecat* (*ltake n xss*)) =
(*ltake* (*eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*))
using *lfusecat-ltake*[*of xss n*] **using** *assms 02 a* **by** *simp*
(*metis Suc-ile-eq gr-implies-not-zero llength-eq-0 order.strict-implies-order*)
have *61*: *llength* (*lfusecat* (*ltake n xss*)) =
llength (*ltake* (*eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*))
using *60* **by** *fastforce*
have *62*: *llength* (*ltake* (*eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*)) =
min (*eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*))

```
      using llength-ltake by blast
    have 63: min (eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
                 (llength (lfusecat xss)) =
              (eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
      using lfusecat-ltake-llength-less-than-llength-lfusecat[of xss  n] a assms
       using Suc-ile-eq order-less-imp-le by simp blast
    show ?thesis
    by (metis 0 60 62 63 assms(3) enat-ord-simps(4) lfinite-ltake lfusecat-lfinite-a
        llength-eq-infty-conv-lfinite lset-ltake subsetD)
  qed
 have 7: 0<n ⟹ eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))) <
       eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))) +
         epred(llength (lnth xss n))
     by (metis 2 6 Suc-ile-eq add.right-neutral assms(4) enat-add-mono less-infinityE
         order-less-imp-le)
 have 8: 0<n ⟹ ?thesis
  using 4 7 eSuc-plus by auto
 show ?thesis
 using 3 8 by blast
qed

lemma ridx-lfusecat-b:
 assumes llastlfirst xss
       ∀ xs ∈ lset xss. 1 < llength xs
       ∀ xs ∈ lset xss. lfinite xs
       ∀ i.  i < llength xss ⟶  ridx R (lnth xss i)
 shows   ridx R (lfusecat xss)
proof −
 have 0: ∀ xs ∈ lset xss. ¬ lnull xs
  using assms(2) gr-implies-not-zero llength-LNil llist.collapse(1) by blast
 have 1: ⋀n. n≤ llength xss ⟹ ridx R (lfusecat (ltake (enat n) xss))
  using assms 0 ridx-lfusecat-ltake-a by blast
 have 2: ⋀n. n≤ llength xss ⟹
       ridx R (ltake (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
                 (lfusecat xss))
  using lfusecat-ltake[of xss ] 1 assms using 0 llist.collapse(1) by fastforce
 have 30: ⋀n . (enat n)≤ llength xss ⟹
           (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i)))) ≤
           llength (lfusecat xss)
      proof −
       fix n::nat
       assume a: n≤ llength xss
       show (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i)))) ≤
            llength (lfusecat xss)
    using lfusecat-ltake-llength-less-than-llength-lfusecat[of xss n ] 0 assms
     a min.order-iff[of (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
       llength (lfusecat xss) ]
     by presburger
   qed
 have 3: ⋀n . (enat n)≤ llength xss ⟹
```

(*if* $n = 0$ *then* 0 *else* $eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i)))) \leq$
llength (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*)))
by (*metis 30 lfusecat-split*)
have *6*: $\bigwedge n.\ (Suc\ n) < llength\ xss \Longrightarrow$
(*if* $n = 0$ *then* 0 *else* $eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i)))) <$
(*if* $(Suc\ n) = 0$ *then* 0 *else* $eSuc(\sum i = 0 .. (n) . epred(llength\ (lnth\ xss\ i))))$
using *assms(1) assms(2) assms(3) assms(4) lfusecat-ltake-llength-less-than-next* **by** *blast*
have *61*: $\bigwedge i.\ i < llength\ xss \Longrightarrow epred(llength\ (lnth\ xss\ i)) < \infty$
by (*metis assms(3) enat-ord-simps(4) epred-0 epred-Infty epred-inject i0-ne-infinity*
in-lset-conv-lnth llength-eq-infty-conv-lfinite)
have *62*: $\bigwedge n.\ n \leq llength\ xss \Longrightarrow$
$(if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i)))) < \infty$
by (*metis* (*no-types, lifting*) *30 6 add-diff-cancel-left′ assms(3) eSuc-enat enat-ord-code(4)*
enat-ord-simps(4) ileI1 leD lfusecat-lfinite-a llength-eq-infty-conv-lfinite plus-1-eq-Suc)
have *7*: $\bigwedge k\ n.\ k \leq (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
$\wedge\ n \leq llength\ xss$
$\Longrightarrow$
ridx R (*ltake* (*enat k*) (*lfusecat xss*))
proof −
fix *k n*
show $k \leq (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
$\wedge\ n \leq llength\ xss$
$\Longrightarrow$*ridx R* (*ltake* (*enat k*) (*lfusecat xss*))
using *ridx-ltake*[*of R* (*if* $n = 0$ *then* 0 *else* $eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
(*lfusecat xss*) *k*]
using *2 30* **by** *presburger*
qed
have *5*: $\bigwedge k.\ k < llength\ xss \Longrightarrow$
ridx R (*ltake* (*enat k*) (*lfusecat xss*))
proof −
fix *k*
show $k < llength\ xss \Longrightarrow$
ridx R (*ltake* (*enat k*) (*lfusecat xss*))
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
by (*metis 1 lfinite-llength-enat linorder-le-cases ltake-all ridx-ltake-a*)
next
case *False*
then show *?thesis*
proof −
have *51*: ¬*lfinite*(*lfusecat xss*)
using *assms 0*
by (*metis False iless-Suc-eq lfusecat-lfinite-b lhd-LCons-ltl*
llength-LCons not-lnull-llength one-enat-def)
have *52*: $\bigwedge k.\ (\exists n.\ enat\ k < (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i)))))$
$\wedge\ n \leq llength\ xss\ \wedge$
ridx R (*ltake* (*if* $n = 0$ *then* 0 *else*
$eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i)))) (lfusecat\ xss)))$

```
    proof −
     fix k
     show (∃ n. enat k < (if n =0 then 0 else eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
                  ∧ n≤llength xss  ∧
                ridx R (ltake (if n =0 then 0 else
                      eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i)))) (lfusecat xss)))
          using 2 6
     proof (induct k)
      case 0
      then show ?case
          proof −
           have 521: (enat 0)< eSuc(∑ i = 0 .. (1−1) . epred(llength (lnth xss i)))
             using i0-iless-eSuc zero-enat-def by presburger
           have 522: (enat 1) ≤ llength xss
             using False
             using lnull-imp-lfinite not-lnull-llength one-enat-def by auto
           have 523: ridx R (ltake (if (enat 1) =0 then 0 else
                      eSuc(∑ i = 0 .. (1−1) . epred(llength (lnth xss i)))) (lfusecat xss))
             using 0.prems(1) 522 one-enat-def by fastforce
           show ?thesis using 521 522 523 one-enat-def by force
          qed
      next
      case (Suc k)
      then show ?case
      by (metis (no-types, lifting) False antisym-conv2 diff-Suc-1 eSuc-enat enat-ord-code(4)
          ileI1 llength-eq-infty-conv-lfinite)
     qed
    qed
   have 53: ⋀k. (∃ m. (enat k) < m ∧  ridx R(ltake (enat m) (lfusecat xss)))
    proof −
     fix k
     show (∃ m. (enat k) < m ∧  ridx R (ltake (enat m) (lfusecat xss)))
     proof −
      have 54: (∃ n.   enat k < (if n =0 then 0 else
                           eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
                  ∧ n≤llength xss  ∧
                 ridx R (ltake (if n =0 then 0 else
                     eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i)))) (lfusecat xss)))
        using 52 by blast
      obtain n where 55: enat k < (if n =0 then 0 else
                               eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i)))) ∧
                     n≤llength xss  ∧
                     ridx R (ltake (if n =0 then 0 else
                           eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i)))) (lfusecat xss))
        using 54 by blast
      show ?thesis
        using 55 62 by force
     qed
    qed
show ?thesis
```

by (*metis 51 53 enat-ord-code(4) llength-eq-infty-conv-lfinite order.strict-implies-order ridx-ltake*)
qed
qed
qed
show *?thesis*
by (*metis 1 5 dual-order.refl enat-ord-code(4) lfinite-conv-llength-enat llength-eq-infty-conv-lfinite ltake-all ridx-ltake-all*)
qed

lemma *ridx-lfusecat*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss. 1* $<$ *llength xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
shows *ridx R* (*lfusecat xss*) $\longleftrightarrow$ ($\forall$ *i.* *i* $<$ *llength xss* $\longrightarrow$ *ridx R* (*lnth xss i*))
using *ridx-lfusecat-a ridx-lfusecat-b assms*
using *gr-implies-not-zero llength-eq-0* **by** *blast*

lemma *lfusecat-split-lsub*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss. 1* $<$ *llength xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
shows ($\forall$ *i. i*$<$ *llength xss* $\longrightarrow$ *ridx* (λ *a b. f* (*lsub a b* σ)) (*lnth xss i*)) $\longleftrightarrow$
ridx (λ *a b. f* (*lsub a b* σ)) (*lfusecat xss*)
using *assms ridx-lfusecat* **by** *blast*

lemma *lidx-lfuse-lfinite*:
assumes *lfinite xs*
llast xs = *lfirst ys*
shows *lidx* (*lfuse xs ys*) $\longleftrightarrow$ *lidx xs* $\wedge$ *lidx ys*
using *assms*
using *ridx-lfuse-lfinite ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ltake*:
assumes *lidx* (*lfusecat xss*)
n $\leq$ *llength xss*
shows *lidx* (*lfusecat* (*ltake n xss*))
using *assms ridx-lfusecat-ltake ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ldrop*:
assumes *lidx* (*lfusecat xss*)
(*enat n*) $<$ *llength xss*
llastlfirst xss
$\forall$ *xs* $\in$ *lset xss.* $\neg$ *lnull xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
shows *lidx* (*lfusecat* (*ldrop n xss*))
using *assms ridx-lfusecat-ldrop ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-a*:

assumes *llastlfirst xss*
$\forall\ xs \in lset\ xss.\ \neg\ lnull\ xs$
$\forall\ xs \in lset\ xss.\ lfinite\ xs$
lidx (*lfusecat xss*)
$i < llength\ xss$
shows *lidx* (*lnth xss i*)
using *assms ridx-lfusecat-a ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-lfuse*:
assumes *llastlfirst xss*
$\forall\ xs \in lset\ xss.\ \neg\ lnull\ xs$
$\forall\ xs \in lset\ xss.\ lfinite\ xs$
$\forall\, i.\ \ i < llength\ xss \longrightarrow\ lidx\ (lnth\ xss\ i)$
$(Suc\ n) < llength\ xss$
shows *lidx* (*lfuse* (*lnth xss n*) (*lnth xss* (*Suc n*)))
using *assms ridx-lfusecat-lfuse ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ltake-a*:
assumes *llastlfirst xss*
$\forall\ xs \in lset\ xss.\ \neg\ lnull\ xs$
$\forall\ xs \in lset\ xss.\ lfinite\ xs$
$\forall\, i.\ \ i < llength\ xss \longrightarrow\ lidx\ (lnth\ xss\ i)$
$n \leq llength\ xss$
shows *lidx* (*lfusecat* (*ltake* (*enat n*) *xss*))
using *assms ridx-lfusecat-ltake-a ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-b*:
assumes *llastlfirst xss*
$\forall\ xs \in lset\ xss.\ 1 < llength\ xs$
$\forall\ xs \in lset\ xss.\ lfinite\ xs$
$\forall\, i.\ \ i < llength\ xss \longrightarrow\ lidx\ (lnth\ xss\ i)$
shows *lidx* (*lfusecat xss*)
using *assms ridx-lfusecat-b ridx-lidx* **by** *blast*

lemma *lidx-lfusecat*:
assumes *llastlfirst xss*
$\forall\ xs \in lset\ xss.\ 1 < llength\ xs$
$\forall\ xs \in lset\ xss.\ lfinite\ xs$
shows *lidx* (*lfusecat xss*) $\longleftrightarrow$ ($\forall\, i.\ \ i < llength\ xss \longrightarrow\ lidx\ (lnth\ xss\ i)$)
using *assms ridx-lfusecat ridx-lidx* **by** *blast*

lemma *lnth-sum-expand*:
assumes $\neg$ *lnull xss*
llastlfirst xss
$\forall\ xs \in lset\ xss.\ 1 < llength\ xs$

$\forall$ *xs* $\in$ *lset xss. lfinite xs*
(*enat i*)< *llength xss*
lfinite xss
shows ($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*))) =
epred(*llength* (*lnth xss i*)) +
($\sum$ *j* $\in$ {*k*. *k*≠*i* $\wedge$ *k*≤ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss j*)))
proof −
have *1*: {*k*. *k*≤ (*the-enat*(*epred*(*llength xss*)))} = *insert i* {*k*. *k*≠*i* $\wedge$ *k*≤ (*the-enat*(*epred*(*llength xss*)))}
using *assms* **by** *auto*
(*metis eSuc-epred enat-ord-simps*(*1*) *epred-enat gr-implies-not-zero iless-Suc-eq*
lfinite-llength-enat the-enat.simps)
have *2*: {*0*.. (*the-enat*(*epred*(*llength xss*)))} = {*k*. *k*≤ (*the-enat*(*epred*(*llength xss*)))}
by *auto*
have *3*: ($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*))) =
($\sum$ *i* ∈{*k*. *k*≤ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss i*)))
using *2* **by** *presburger*
have *4*: ($\sum$ *i* ∈{*k*. *k*≤ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss i*))) =
($\sum$ *j* ∈(*insert i* {*k*. *k*≠*i* $\wedge$ *k*≤ (*the-enat*(*epred*(*llength xss*)))}) . *epred*(*llength* (*lnth xss j*)))
using *1* **by** *presburger*
have *5*: ($\sum$ *j* ∈(*insert i* {*k*. *k*≠*i* $\wedge$ *k*≤ (*the-enat*(*epred*(*llength xss*)))}) . *epred*(*llength* (*lnth xss j*))) =
epred(*llength* (*lnth xss i*)) +
($\sum$ *j* $\in$ {*k*. *k*≠*i* $\wedge$ *k*≤ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss j*)))
by *auto*
show *?thesis*
using *3 4 5* **by** *presburger*
qed

lemma *lfusecat-llength-a*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss. 1* < *llength xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
i < *llength xss*
(*enat j*) < *llength* (*lnth xss i*)
shows (*enat j*) < *llength* (*lfusecat xss*)
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
proof −
have *1*: *lnull xss* $\Longrightarrow$ *?thesis*
using *assms*(*4*) *gr-implies-not-zero llength-eq-0* **by** *blast*
have *2*: ¬ *lnull xss* $\Longrightarrow$
llength (*lfusecat xss*) =
eSuc($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*)))
using *True assms lfusecat-llength-lfinite* **by** *fastforce*
have *3*: ¬ *lnull xss* $\Longrightarrow$
llength (*lnth xss i*) ≤
eSuc($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*)))
using *lnth-sum-expand*[*of xss i*] *assms*
by (*metis* (*no-types*, *lifting*) *True co.enat.collapse eSuc-plus gr-implies-not-zero le-iff-add*)

```
  have 4: ¬ lnull xss ⟹ ?thesis
    using 2 3
    by (metis assms(5) order-less-le-trans)
  show ?thesis
  using 1 4 by blast
 qed
next
case False
then show ?thesis
 proof −
  have 5: ¬ lfinite (lfusecat xss)
    by (metis False assms(1) assms(2) assms(3) gr-implies-not-zero iless-Suc-eq lfusecat-lfinite-b
        lhd-LCons-ltl llength-LCons llength-eq-0 not-lnull-llength one-enat-def)
  show ?thesis
  using 5 enat-iless lfinite-conv-llength-enat not-less-iff-gr-or-eq by blast
 qed
qed

lemma lmap-lfusecat:
 lmap f (lfusecat xss) = (lfusecat (lmap ( lmap f ) xss))
proof (induct xss)
case adm
then show ?case by simp
next
case LNil
then show ?case
by simp
next
case (LCons xs xss)
then show ?case
by (simp add: lfuse-def  )
  (metis lmap-eq-LNil lmap-lappend-distrib lnull-def ltl-lmap)
qed



lemma lfusecat-is-lfirst-conv:
 assumes ∀ i. i< llength xss ⟶ is-lfirst(lnth xss i)
         is-lfirst xss
 shows   is-lfirst(lfusecat xss)
using assms
proof (induction xss)
case adm
 then show ?case
  by (rule ccpo.admissibleI)  (auto, metis lnth-0 zero-enat-def)
next
case LNil
then show ?case by simp
next
case (LCons xs xss)
```

then show *?case*
by (*simp add*: *zero-enat-def*)
qed

lemma *lconcat-split-llast*:
assumes *lfinite ll*
shows *lconcat ll* = (*if lnull ll then LNil else lappend* (*lconcat* (*lbutlast ll*)) (*llast ll*))
using *assms*
proof (*induction ll*)
case *lfinite-LNil*
then show *?case* **by** *simp*
next
case (*lfinite-LConsI xs x*)
then show *?case* **apply** *auto*
apply (*metis lappend-code*(*1*) *lbutlast-simps*(*2*) *llast-singleton lnull-def*)
by (*metis lappend-assoc lbutlast-simps*(*3*) *lconcat-LCons llast-LCons not-lnull-conv*)
qed

lemma *lfusecat-lconcat-lmap-lbutlast-lfinite*:
assumes *lfinite ll*
⋀ *xx*. *xx* ∈ *lset ll* ⟹ *llength xx* >*1*
⋀ *xx*. *xx* ∈ *lset ll* ⟹ *lfinite xx*
llastlfirst ll
shows *lfusecat ll* =
(*if lnull ll then LNil else*
(*lappend* (*lconcat* (*lbutlast* (*lmap* (*lbutlast*) *ll*))) (*llast ll*)))
using *assms*
proof (*induction ll*)
case *lfinite-LNil*
then show *?case* **by** *simp*
next
case (*lfinite-LConsI xs x*)
then show *?case*
proof (*cases lnull xs*)
case *True*
then show *?thesis* **using** *lfinite-LConsI*
by (*metis lappend-lnull1 lbutlast-simps*(*2*) *lconcat-LNil lfuse-LNil-2 lfusecat-LCons lfusecat-LNil llast-singleton llist.simps*(*12*) *lmap-lbutlast*
lnull-def)
next
case *False*
then show *?thesis*
proof −
have *1*: *lfusecat* (*LCons x xs*) = *lfuse x* (*lfusecat xs*)
by *simp*
have *2*: (*if lnull* (*LCons x xs*) *then LNil else lappend* (*lconcat* (*lbutlast* (*lmap lbutlast* (*LCons x xs*)))) (*llast* (*LCons x xs*))) =
(*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast* (*LCons x xs*)))) (*llast* (*LCons x xs*)))
by *simp*

have *3*: (*llast* (*LCons x xs*)) = *llast xs*
by (*simp add*: *False llast-LCons*)
have *4*: *lfinite xs*
by (*simp add*: *lfinite-LConsI.hyps*)
have *5*: (⋀*xx*. *xx* ∈ *lset xs* ⟹ *1* < *llength xx*)
by (*simp add*: *lfinite-LConsI.prems*(*1*))
have *6*: (⋀*xx*. *xx* ∈ *lset xs* ⟹ *lfinite xx*)
by (*simp add*: *lfinite-LConsI.prems*(*2*))
have *7*: *llastlfirst xs*
using *False lfinite-LConsI.prems*(*3*) **by** *auto*
have *8*: *lfusecat xs* = (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast xs*))) (*llast xs*))
using *5 6 7 lfinite-LConsI.IH False* **by** *presburger*
have *9*: *lfusecat* (*LCons x xs*) =
lfuse x (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast xs*))) (*llast xs*))
by (*simp add*: *8*)
have *10*: (*if lnull* (*LCons x xs*) *then LNil else lappend* (*lconcat* (*lbutlast* (*lmap lbutlast* (*LCons x xs*))))
(*llast* (*LCons x xs*))) =
(*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast* (*LCons x xs*)))) (*llast* (*xs*)))
by (*simp add*: *3*)
have *11*: *llast x* = *lfirst* (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast xs*))) (*llast xs*))
by (*metis 5 8 False lfinite-LConsI.prems*(*3*) *lfirst-def lfirst-lfusecat-lfirst llastlfirst-LCons llength-LNil*
llist.set-sel(*1*) *lnull-def*
not-one-less-zero)
have *12*: (*lmap lbutlast* (*LCons x xs*)) = (*LCons* (*lbutlast x*) (*lmap lbutlast xs*))
by *simp*
have *13*: (*lbutlast* (*LCons* (*lbutlast x*) (*lmap lbutlast xs*))) =
(*LCons* (*lbutlast x*) (*lbutlast* (*lmap lbutlast xs*)))
by (*simp add*: *False lbutlast-eq-LCons-conv llast-LCons*)
have *14*: (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast* (*LCons x xs*)))) (*llast* (*xs*))) =
(*lappend* (*lappend* (*lbutlast x*) (*lconcat* (*lbutlast* (*lmap lbutlast xs*)))) (*llast* (*xs*)))
by (*simp add*: *13*)
have *15*: (*lappend* (*lappend* (*lbutlast x*) (*lconcat* (*lbutlast* (*lmap lbutlast xs*)))) (*llast* (*xs*))) =
(*lappend* (*lbutlast x*) (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast xs*))) (*llast* (*xs*))))
by (*simp add*: *lappend-assoc*)
have *16*: *lfuse x* (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast xs*))) (*llast xs*)) =
(*lappend* (*lbutlast x*) (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast xs*))) (*llast* (*xs*))))
by (*metis 11 5 8 False lfuse-lbutlast lfusecat-not-lnull-var nless-le not-lnull-llength*)
show *?thesis*
using *14 15 16 3 8* **by** *auto*
qed
qed
qed

lemma *lfusecat-lconcat-lmap-lbutlast*:
assumes ⋀ *xx*. *xx* ∈ *lset ll* ⟹ *llength xx* >*1*
⋀ *xx*. *xx* ∈ *lset ll* ⟹ *lfinite xx*
llastlfirst ll
¬ *lfinite ll*
shows *lfusecat ll* = *lconcat* (*lmap* (*lbutlast*) *ll*)

using *assms*
proof (*coinduction arbitrary*: *ll*)
case (*Eq-llist ll1*)
then show *?case*
proof −
have *1*: ∃ *x xs*. *ll1* = (*LCons x xs*)
using *Eq-llist*(*4*) *llist.exhaust-sel* **by** *blast*
obtain *x xs* **where** *01*: *ll1* = (*LCons x xs*)
using *1* **by** *blast*

have *02*: *llast x* = *lfirst* (*lfirst xs*)
using *Eq-llist 01*
by (*meson lfinite-code*(*2*) *llastlfirst-LCons lnull-imp-lfinite*)
have *2*: *lnull* (*lfusecat ll1*) = *lnull* (*lconcat* (*lmap lbutlast ll1*))
using *01 Eq-llist*(*1*)
by (*metis* (*no-types*, *lifting*) *co.enat.exhaust-sel iless-Suc-eq is-LMore-llength lappend.disc*(*2*) *lconcat-LCons*
lfusecat-not-lnull llength-eq-0 llength-lbutlast llist.simps(*13*) *lset-intros*(*1*) *not-lnull-conv*
not-lnull-llength one-enat-def)
have *3*: (¬ *lnull* (*lfusecat ll1*) ⟶
¬ *lnull* (*lconcat* (*lmap lbutlast ll1*)) ⟶
lhd (*lfusecat ll1*) = *lhd* (*lconcat* (*lmap lbutlast ll1*))) **using** *1*
apply *auto*
apply (*metis lbutlast-conv-ltake llength-eq-0 llength-ltl ltake.simps*(*3*))
using *Eq-llist*(*1*)
by (*metis gr-implies-not-zero is-LEmpty-llength lbutlast-conv-ltake lhd-LCons linorder-neq-iff llength-eq-0*
llength-ltl llist.exhaust-sel lnull-def lset-intros(*1*) *ltake.code*)
have *4*: ¬ *lnull* (*lfusecat ll1*)
by (*metis Eq-llist*(*1*,*4*) *lfinite-LNil lfusecat-not-lnull-var*
linorder-not-less llength-lnull lnull-def
zero-le)
have *5*: ¬ *lnull* (*lconcat* (*lmap lbutlast ll1*))
using *2 4* **by** *blast*
have *6*: *ltl* (*lfusecat* (*LCons x xs*)) = *ltl* (*lfuse x* (*lfusecat xs*))
by *simp*
have *7*: *llength x* > *1*
by (*simp add*: *01 Eq-llist*(*1*))
have *8*: *ltl* (*lfuse x* (*lfusecat xs*)) = *lfuse* (*ltl x*) (*lfusecat xs*)
using *7* **apply** *simp* **unfolding** *lfuse-def* **apply** *simp*
by (*metis lappend-lnull2 lhd-LCons-ltl llength-LCons*
llength-LNil llist.collapse(*1*) *nless-le not-lnull-llength*
one-eSuc)
have *9*: *ltl* (*lconcat* (*lmap lbutlast* (*LCons x xs*))) = *ltl* (*lappend* (*lbutlast x*) (*lconcat* (*lmap lbutlast xs*)))

by *simp*
have *10*: *ltl* (*lappend* (*lbutlast x*) (*lconcat* (*lmap lbutlast xs*))) =
lappend (*ltl* (*lbutlast x*)) (*lconcat* (*lmap lbutlast xs*))
by (*metis 7 is-LEmpty-llength lappend-ltl lbutlast.ctr*(*1*)
lbutlast.disc-iff(*2*) *lbutlast-eq-LNil-conv nless-le*
not-lnull-llength)
have *11*: (*ltl* (*lbutlast x*)) = *lbutlast* (*ltl x*)

by (*simp add*: *lbutlast-ltl*)
have *12*: *lfuse* (*ltl x*) (*lfusecat xs*) = *lfusecat* (*LCons* (*ltl x*) *xs*)
by *simp*
have *13*: *lappend* (*ltl* (*lbutlast x*)) (*lconcat* (*lmap lbutlast xs*)) =
lconcat (*lmap lbutlast* (*LCons* (*ltl x*) *xs*))
by (*simp add*: *lbutlast-ltl*)
have *14*: *llength x* = *2* ⟹ *llast* (*ltl x*) = *lfirst* (*lfusecat xs*)
using *02*
using *01 Eq-llist*(*1*,*4*)
by (*metis eq-LConsD insert-iff is-LMore-llength lfinite-LNil lfinite-code*(*2*) *lfirst-def lfirst-lfusecat-lfirst llast-LCons2*
llist.collapse(*1*,*2*) *llist.set*(*2*))
have *15*: *llength x* = *2* ⟹ *lfuse* (*ltl x*) (*lfusecat xs*) = (*lfusecat xs*)
using *14*
by (*metis 01 Eq-llist*(*1*,*4*) *co.enat.sel*(*2*) *epred-simps*(*2*) *is-LMore-llength lfinite-ltl lfirst-def lfuse-One-a lfusecat-not-lnull-var lhd-LCons-ltl llast-LCons*
llength-eq-0 llength-ltl llist.collapse(*1*) *llist.disc*(*2*) *lnull-imp-lfinite lset-intros*(*2*) *ltl-simps*(*2*) *one-add-one plus-1-eSuc*(*2*) *zero-neq-one*)
have *16*: *llength x* = *2* ⟹ *lappend* (*ltl* (*lbutlast x*)) (*lconcat* (*lmap lbutlast xs*)) = (*lconcat* (*lmap lbutlast xs*))
by (*metis diff-Suc-1 epred-simps*(*2*,*6*) *lappend-lnull1 llength-lbutlast llength-ltl not-lnull-llength not-one-le-zero one-add-one one-enat-def plus-1-eq-Suc*
plus-enat-simps(*1*))
have *17*: (∀ *xx*. *xx* ∈ *lset xs* ⟶ *1* < *llength xx*)
by (*simp add*: *01 Eq-llist*(*1*))
have *18*: (∀ *xx*. *xx* ∈ *lset xs* ⟶ *lfinite xx*)
by (*simp add*: *01 Eq-llist*(*2*))
have *19*: *llastlfirst xs*
using *01 Eq-llist*(*3*,*4*) *llastlfirst-LCons lnull-imp-lfinite* **by** *auto*
have *20*: ¬ *lfinite xs*
using *01 Eq-llist*(*4*) **by** *auto*
have *21*: *llength x* > *2* ⟹ (∀ *xx*. *xx* ∈ *lset* (*LCons* (*ltl x*) *xs*) ⟶ *1* < *llength xx*)
by (*metis 17 7 Extended-Nat.eSuc-mono eSuc-enat enat-defs*(*2*) *enat-numeral insert-iff less-imp-le lhd-LCons-ltl llength-LCons lset-LCons nat-1-add-1*
not-lnull-llength plus-1-eq-Suc)
have *22*: (∀ *xx*. *xx* ∈ *lset* (*LCons* (*ltl x*) *xs*) ⟶ *lfinite xx*)
using *01 Eq-llist*(*2*) **by** *auto*
have *23*: ¬ *lfinite* (*LCons* (*ltl x*) *xs*)
by (*simp add*: *20*)
have *24*: *llength x* > *2* ⟹ *?thesis*
by (*metis 01 02 10 12 13 19 20 21 22 23 3 4 5 6 7 8 9 is-LMore-llength llast-LCons llastlfirst-LCons lnull-imp-lfinite*
ltl-simps(*2*))
have *25*: *llength x* = *2* ⟹ *?thesis*
by (*metis 01 10 15 16 17 18 19 20 3 4*
5 6 8 9)
show *?thesis*
by (*metis 24 25 7 ileI1 nless-le one-add-one plus-1-eSuc*(*2*))
qed
qed

1.1.8 kfilter

lemma *kfilter-code* [*simp*, *code*]:
shows *kfilter-LNil*: *kfilter P n LNil* = *LNil*
and *kfilter-LCons*: *kfilter P n* (*LCons x xs*) =
(*if P x then LCons n* (*kfilter P* (*Suc n*) *xs*) *else kfilter P* (*Suc n*) *xs*)
by (*auto simp add*: *kfilter-def lzip.ctr*(*2*))

lemma *lmap-fst-imp-a*:
assumes *lnull* (*lfilter P xs*)
shows *lnull* (*lmap fst* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))
using *assms lset-lzipD1* **by** *fastforce*

lemma *lmap-fst-imp-b*:
assumes *lnull* (*lmap fst* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))
shows *lnull* (*lfilter P xs*)
proof −
have *0*: *lnull* (*lmap fst* (*lfilter* (λ (*x*,*k*). *P x*) (*lzip xs* (*iterates Suc n*))))
using *assms* **by** *auto*
have *1*: *lnull* (*lfilter* (λ (*x*,*k*). *P x*) (*lzip xs* (*iterates Suc n*)))
using *0* **by** *auto*
have *2*: (∀ (*x*,*k*) ∈ *lset*(*lzip xs* (*iterates Suc n*)). ¬ *P x*)
using *1* **by** (*simp add*: *prod.case-eq-if*)
have *3*: (∀ (*x*,*k*) ∈ { (*lnth xs i* , *n*+*i*) | *i*. *enat i* < *llength xs*}. ¬ *P x*)
using *2* **by** (*simp add*: *lset-lzip*)
have *4*: (∀ *x* ∈ { *lnth xs i* | *i*. *enat i* < *llength xs*}. ¬ *P x*)
using *3* **by** *blast*
from *4* **show** *?thesis* **by** (*simp add*: *lset-conv-lnth*)
qed

lemma *lmap-snd-lnull*:
lnull (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))) = *lnull*(*lfilter P xs*)
by (*metis llist.collapse*(*1*) *llist.disc*(*1*) *lmap-eq-LNil lmap-fst-imp-a lmap-fst-imp-b*)

lemma *kfilter-lnull-conv*:
lnull (*kfilter P n xs*) ⟷ (∀ *x* ∈ *lset xs*. ¬ *P x*)
unfolding *kfilter-def* **using** *lmap-snd-lnull*[*of P xs*] *lnull-lfilter*[*of P xs*] **by** *blast*

lemma *kfilter-not-lnull-conv*:
¬*lnull* (*kfilter P n xs*) ⟷ (∃ *x* ∈ *lset xs*. *P x*)
by (*simp add*: *kfilter-lnull-conv*)

lemma *lmap-fst-lfilter*:
lfilter P (*lmap fst* (*lzip xs* (*iterates Suc n*))) =
lmap fst (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))
using *lfilter-lmap* **by** *blast*

lemma *lmap-fst-lzip-a*:
assumes *llength xs* = *llength ys*
shows *lmap fst* (*lzip xs ys*) = *xs*
using *assms*

proof (*coinduction arbitrary*: *xs ys*)
case *Eq-llist*
then show *?case*
by *simp*
(*metis epred-llength llength-eq-0*)
qed

lemma *lmap-snd-lzip*:
assumes *llength xs* = *llength ys*
shows *lmap snd* (*lzip xs ys*) = *ys*
using *assms*
proof (*coinduction arbitrary*: *xs ys*)
case *Eq-llist*
then show *?case*
by *simp*
(*metis epred-llength llength-eq-0*)
qed

lemma *lmap-fst-lzip*:
(*lmap fst* (*lzip xs* (*iterates Suc n*))) = *xs*
by (*coinduction arbitrary*: *xs*) (*auto*, *simp add*: *lmap-fst-lzip-conv-ltake*)

lemma *lfilter-kfilter-1*:
(*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
(*lfilter P xs*)
by (*metis* (*mono-tags*, *lifting*) *lmap-fst-lfilter lmap-fst-lzip*)

lemma *lfilter-kfilter-snd-llength*:
llength (*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
llength (*lmap snd* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
by *simp*

lemma *kfilter-llength*:
llength(*kfilter P n xs*) = *llength*(*lfilter P xs*)
proof −
have *1*: *llength*(*kfilter P n xs*) = *llength* (*lmap snd* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
by (*simp add*: *kfilter-def*)
have *2*: *llength* (*lmap snd* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
llength (*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
using *lfilter-kfilter-snd-llength* **by** *auto*
have *3*: *llength* (*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
llength(*lfilter P xs*)
by (*metis lfilter-kfilter-1*)
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *ldropWhile-LEAST*:
assumes ∃ *n*<*llength xs*. (*Not* ∘ *P*) (*lnth xs n*)
shows *ldropWhile P xs* = *ldropn* (*LEAST n*. *n* < *llength xs* ∧ (*Not* ∘*P*) (*lnth xs n*)) *xs*
proof −

from *assms* **obtain** *m* **where**
m< *llength xs* ∧ (*Not* ∘ *P*) (*lnth xs m*)
⋀*n*. *n*< *llength xs* ⟶ (*Not* ∘*P*) (*lnth xs n*) ⟹ *m* ≤ *n*
and ∗: (*LEAST n*. *n* < *llength xs* ∧ (*Not* ∘ *P*) (*lnth xs n*)) = *m*
by *atomize-elim*
(*metis* (*no-types*, *lifting*) *dual-order.strict-trans1 enat-ord-code*(*2*) *less-imp-le not-le-imp-less*
not-less-Least wellorder-Least-lemma(*1*))
thus *?thesis* **unfolding** ∗
proof (*induct m arbitrary*: *xs*)
case *0*
then show *?case*
by (*cases xs*) *simp-all*
next
case (*Suc m*)
then show *?case*
proof −
have *1*: *ldropWhile P* (*ltl xs*) = *ldropn m* (*ltl xs*) **using** *Suc*
by (*cases xs*)
(*simp*,
metis Suc-le-mono ldrop-eSuc-ltl ldropn-Suc-conv-ldropn ldropn-eq-LNil llist.simps(*3*)
lnth-Suc-LCons ltl-simps(*2*) *not-le-imp-less*)
have *2*: *P* (*lnth xs 0*)
using *Suc.prems*(*2*) **by** *auto*
show *?thesis*
by (*metis 1 2 ldropWhile-LCons ldrop-eSuc-ltl lhd-LCons-ltl llist.collapse*(*1*)
lnth-0-conv-lhd ltl-simps(*1*))
qed
qed
qed

lemma *ldropWhile-LEAST-not*:
assumes ∃ *n*<*llength xs*. *P* (*lnth xs n*)
shows *ldropWhile* (*Not* ∘ *P*) *xs* = *ldropn* (*LEAST n*. *n* < *llength xs* ∧ *P* (*lnth xs n*)) *xs*
using *assms ldropWhile-LEAST*[*of xs Not* ∘ *P*] **by** *simp*

lemma *ltakeWhile-LEAST*:
assumes ∃ *n*<*llength xs*. (*Not* ∘ *P*) (*lnth xs n*)
shows (*ltakeWhile P xs*) = (*ltake* (*lleast* (*Not* ∘ *P*) *xs*) *xs*)
proof−
from *assms* **obtain** *m* **where**
m< *llength xs* ∧ (*Not* ∘ *P*) (*lnth xs m*)
⋀*n*. *n*< *llength xs* ⟶ (*Not* ∘*P*) (*lnth xs n*) ⟹ *m* ≤ *n*
and ∗: (*lleast* (*Not* ∘ *P*) *xs*) = *m* **unfolding** *lleast-def*
by *atomize-elim*
(*metis* (*no-types*, *lifting*) *Least-le dual-order.trans enat-ord-code*(*2*) *not-less not-less-iff-gr-or-eq*
wellorder-Least-lemma(*1*))
thus *?thesis* **unfolding** ∗
proof (*induct m arbitrary*: *xs*)
case *0*
then show *?case*

proof −
have *ltakeWhile P* (*LCons* (*lnth xs 0*) (*ldropn* (*Suc 0*) *xs*)) = *LNil*
using *0* **by** *fastforce*
then show *?thesis*
by (*metis* (*no-types*) *lappend.disc-iff*(*2*) *lappend-ltakeWhile-ldropWhile ldropn-0*
ldropn-Suc-conv-ldropn llist.collapse(*1*) *lnull-ldropn ltake.ctr*(*1*) *not-less zero-enat-def*)
qed
next
case (*Suc m*)
then show *?case*
proof −
have *1*: *ltakeWhile P* (*ltl xs*) = *ltake m* (*ltl xs*)
using *Suc* **by** (*cases xs*)
(*simp*,
metis Extended-Nat.eSuc-mono Suc-le-mono eSuc-enat llength-LCons lnth-Suc-LCons ltl-simps(*2*))
have *2*: *P* (*lnth xs 0*)
using *Suc* **by** *auto*
show *?thesis*
by (*metis 1 2 eSuc-enat lhd-LCons-ltl lnth-0-conv-lhd ltake.ctr*(*1*) *ltakeWhile.code*
ltake-eSuc-LCons)
qed
qed
qed

lemma *llength-LEAST-not*:
assumes ∃ *n* < *llength xs*. (*Not* ∘ *P*) (*lnth xs n*)
shows (*lleast* (*Not* ∘ *P*) *xs*) < *llength xs*
using *assms* **unfolding** *lleast-def*
by (*metis* (*no-types*, *lifting*) *LeastI*)

lemma *llength-LEAST*:
assumes ∃ *n* < *llength xs*. *P* (*lnth xs n*)
shows (*lleast P xs*) < *llength xs*
using *assms llength-LEAST-not*[*of xs Not* ∘ *P*] **unfolding** *lleast-def* **by** *simp*

lemma *llength-ltakeWhile-LEAST*:
assumes ∃ *n*< *llength xs*. (*Not* ∘ *P*) (*lnth xs n*)
shows (*llength* (*ltakeWhile P xs*)) = (*lleast* (*Not* ∘ *P*) *xs*)
using *assms ltakeWhile-LEAST*[*of xs P*] **unfolding** *lleast-def*
by (*metis* (*no-types*, *lifting*) *dual-order.strict-trans1 enat-ord-simps*(*2*) *le-cases llength-ltake*
min-def not-less-Least)

lemma *llength-ltakeWhile-LEAST-not*:
assumes ∃ *n*< *llength xs*. *P* (*lnth xs n*)
shows (*llength* (*ltakeWhile* (*Not* ∘*P*) *xs*)) = (*lleast P xs*)
using *assms llength-ltakeWhile-LEAST*[*of xs Not* ∘ *P*] **unfolding** *lleast-def* **by** *simp*

lemma *lzip-ldropWhile-fst*:
assumes *llength* (*lzip xs ys*) = *llength xs*
shows *lzip* (*ldropWhile P xs*) (*lmap snd* (*ldropWhile* (*P* ∘ *fst*) (*lzip xs ys*))) =

ldropWhile (*P* ∘ *fst*) (*lzip xs ys*)
using *assms*
proof −
have *f1*: *ldropWhile P xs* = *lmap fst* (*ldropWhile* (*P* ∘ *fst*) (*lzip xs ys*))
by (*metis* (*no-types*) *llength-lzip assms ldropWhile-lmap lmap-fst-lzip-conv-ltake ltake-all order-refl*)
have *ltake* (*min* (*llength* (*ldrop* (*llength* (*ltakeWhile* (*P* ∘ *fst*) (*lzip xs ys*))) *xs*))
(*llength* (*ldrop* (*llength* (*ltakeWhile* (*P* ∘ *fst*) (*lzip xs ys*))) *ys*)))
(*ldrop* (*llength* (*ltakeWhile* (*P* ∘ *fst*) (*lzip xs ys*))) (*lzip xs ys*)) =
ldrop (*llength* (*ltakeWhile* (*P* ∘ *fst*) (*lzip xs ys*))) (*lzip xs ys*)
by (*simp add*: *ltake-all*)
then show *?thesis*
using *f1* **by** (*simp add*: *ldropWhile-eq-ldrop lmap-fst-lzip-conv-ltake lmap-snd-lzip-conv-ltake ltake-lzip*)
qed

lemma *lzip-ldropWhile-fst-iterates*:
ldropWhile (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)) =
lzip (*ldropWhile* (*Not* ∘*P*) *xs*) (*lmap snd* (*ldropWhile* (*Not* ∘*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))
by (*metis llength-lmap lmap-fst-lzip lzip-ldropWhile-fst*)

lemma *ldropWhile-iterates-split*:
assumes ∃ *n*<*llength xs*. *P* (*lnth xs n*)
shows ((*ldropWhile* (*Not* ∘*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))) =
(*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)
(*iterates Suc* (*n*+(*lleast P xs*))))
proof −
have *1*: ∃ *na*. *enat na* < *llength* (*lzip xs* (*iterates Suc n*)) ∧
(*Not*∘ (*Not*∘ *P*) ∘ *fst*) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)
using *lmap-fst-lzip*[*of xs n*]
by (*metis assms comp-apply llength-lmap lnth-lmap*)
have *2*: (*ldropWhile* ((*Not* ∘*P*) ∘ *fst*) (*lzip xs* (*iterates Suc n*))) =
(*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
((*Not* ∘ (*Not* ∘*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*)))
using *1 ldropWhile-LEAST*[*of* (*lzip xs* (*iterates Suc n*)) (*Not* ∘*P*) ∘ *fst*] **by** *auto*
have *3*: (*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
((*Not* ∘ (*Not* ∘*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*))) =
(*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*)))
by *auto*
have *4*: (*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*))) =
(*lzip* (*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) *xs*)
(*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*iterates Suc n*)))
using *ldropn-lzip* **by** *blast*
have *5*: (*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*iterates Suc n*)) =
(*iterates Suc* (*n*+(*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*))))
by (*metis* (*no-types*, *lifting*) *funpow-Suc-conv ldropn-iterates*)

have *6*: (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) =
(*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P* ∘*fst*))) (*lnth* (*xs*) *na*, *lnth* (*iterates Suc n*) *na*))
by (*metis* (*no-types*, *opaque-lifting*) *comp-def fst-conv lmap-fst-lzip lnth-lmap*)
have *7*: *llength*(*lzip xs* (*iterates Suc n*)) = *llength xs*
by *simp*
have *8*: (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P* ∘*fst*))) (*lnth* (*xs*) *na*, *lnth* (*iterates Suc n*) *na*)) =
(*LEAST na. na* < *llength xs* ∧ *P* (*lnth xs na*))
by *auto*
have *9*: (*lzip* (*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) *xs*)
(*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*iterates Suc n*))) =
(*lzip* (*ldropn* (*LEAST na. na* < *llength xs* ∧ *P* (*lnth xs na*)) *xs*)
(*iterates Suc* (*n*+(*LEAST na. na* < *llength xs* ∧ *P* (*lnth xs na*)))))
using *5 6* **by** *auto*
show *?thesis* **unfolding** *lleast-def*
using *assms 2 3 4 9 ldropWhile-LEAST-not*[*of xs P*] **by** *presburger*
qed

lemma *kfilter-ldropWhile* :
assumes ¬ *lnull*(*kfilter P n xs*)
shows *kfilter P n xs* =
(*LCons* (*n*+ (*lleast P xs*))
(*lmap snd* (*lfilter* (*P*∘*fst*)
(*lzip* (*ldrop* (*Suc* (*lleast P xs*)) (*xs*))
(*iterates Suc* (*Suc* (*n*+(*lleast P xs*))))))))
proof −
let *?Least* = (*lleast P xs*)
have *1*: *lhd*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))) =
lhd (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))
by *simp*
have *2*: *ltl*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))) =
(*lfilter* (*P*∘*fst*) (*ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))))
by (*simp add*: *ltl-lfilter*)
have *3*: *lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)) =
(*LCons* (*lhd*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
(*ltl*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))))
by (*metis assms kfilter-llength llength-eq-0 llength-lmap llist.disc*(*1*) *llist.exhaust-sel*
lmap-fst-lfilter lmap-fst-lzip)
have *4*: *kfilter P n xs* =
lmap snd (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))
by (*simp add*: *kfilter-def*)
have *5*: *ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))) =
ltl((*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)
(*lmap snd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))))
by (*metis lzip-ldropWhile-fst-iterates*)
have *6*: *ltl*((*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)

```
                (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))))) =
      (lzip (ltl (ldropWhile (Not ∘P) xs))
          (ltl (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))))
  using lzip-ldropWhile-fst-iterates[of P xs n]
      ltl-lzip[of (ldropWhile (Not ∘ P) xs)
                (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))]
  by force
have 7: lmap snd (
      (LCons (lhd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))
          (lfilter (P∘fst) (ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))))) =
       (LCons (snd (lhd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))))
          (lmap snd (lfilter (P∘fst) (ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))))))
  by simp
have 8: ∃ n. enat n < llength xs ∧ P (lnth xs n)
   by (metis assms in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter)
have 9: (snd (lhd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))) =
       (snd (lhd (lzip  (ldropWhile (Not ∘P) xs)
          (iterates Suc (n+?Least)) )))
  using 8 ldropWhile-iterates-split[of xs P n] by simp
have 10: (snd (lhd (lzip  (ldropWhile (Not ∘P) xs)
          (iterates Suc (n+?Least)) ))) =
        lhd (iterates Suc (n+?Least))
  by (metis (no-types, lifting) 3 comp-assoc iterates.disc-iff lfilter-eq-LCons lhd-lzip
     llist.disc(2) lzip.disc(1) lzip-ldropWhile-fst-iterates snd-conv)
have 11: lhd (iterates Suc (n+?Least)) = (n+?Least)
  by simp
have 12: ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
       ltl (lzip  (ldropWhile (Not ∘P) xs)
          (iterates Suc (n+?Least)) )
  using 8 ldropWhile-iterates-split[of xs P n] by simp
have 13: ltl (lzip  (ldropWhile (Not ∘P) xs) (iterates Suc (n+?Least)) ) =
       (lzip (ltl (ldropWhile (Not ∘P) xs)) (ltl (iterates Suc (n+?Least))))
  by (metis (no-types, lifting) 3 comp-assoc iterates.disc-iff lfilter-eq-LCons llist.discI(2)
      ltl-lzip lzip.disc-iff(2) lzip-ldropWhile-fst-iterates)
have 14: (ltl (iterates Suc (n+?Least))) = (iterates Suc (Suc (n+?Least)))
  by simp
have 15: ltl (ldropWhile (Not ∘P) xs) =  ltl (ldrop (llength (ltakeWhile (Not ∘P) xs)) xs)
  by (simp add: ldropWhile-eq-ldrop)
have 16: ltl (ldrop (llength (ltakeWhile (Not ∘P) xs)) xs) =
        ldrop (eSuc (llength (ltakeWhile (Not ∘P) xs))) ( xs)
  by (simp add: ldrop-eSuc-conv-ltl)
have 17: (llength (ltakeWhile (Not ∘P) xs)) = (llength (ltake ?Least xs))
  using 8 ltakeWhile-LEAST[of xs Not ∘P ] unfolding lleast-def by auto
have 18: (llength (ltake ?Least xs)) = enat ?Least
  using 17 8 llength-ltakeWhile-LEAST-not unfolding lleast-def by fastforce
have 19: ldrop (eSuc (llength (ltakeWhile (Not ∘P) xs))) ( xs) = ldrop (Suc ?Least) ( xs)
  using 17 18 by (simp add: eSuc-enat)
have 20: ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
        (lzip (ldrop (Suc ?Least) ( xs)) (iterates Suc (Suc (n+?Least))))
  using 12 13 15 16 19  by auto
```

show *?thesis*
using *2 3 4 10 20 9* **by** *auto*
qed

lemma *kfilter-eq-LCons*:
kfilter P n xs = LCons x xs′ ⟹
x = n+(lleast P xs) ∧
xs′ = (lmap snd (lfilter (P∘fst)
(lzip (ldrop (Suc (lleast P xs)) (xs))
(iterates Suc (Suc (n+(lleast P xs)))))))
using *kfilter-ldropWhile*[*of P n xs*] **by** *auto*

lemma *kfilter-eq-LCons-1*:
kfilter P n xs = LCons x xs′ ⟹
x = (n+(lleast P xs)) ∧
xs′ = kfilter P (Suc (n+(lleast P xs))) (ldrop (Suc (lleast P xs)) xs)
using *kfilter-eq-LCons*[*of P n xs x xs′*]
kfilter-def[*of P (Suc (n + (lleast P xs)))*
(ldrop (Suc (lleast P xs)) xs)]
by *auto*

lemma *kfilter-eq-conv*:
kfilter P n xs = LNil ∨
kfilter P n xs =
LCons (n+(lleast P xs)) (kfilter P (Suc (n+(lleast P xs))) (ldrop (Suc (lleast P xs)) xs))
proof (*cases kfilter P n xs*)
case *LNil*
then show *?thesis* **by** *simp*
next
case (*LCons x21 x22*)
then show *?thesis*
proof −
let *?Least* = (*lleast P xs*)
have *1*: *x21 = n+?Least*
using *kfilter-eq-LCons-1*[*of P n xs x21 x22*] **by** (*meson LCons*)
have *2*: *x22*= (*kfilter P* (*Suc* (*n+?Least*)) (*ldrop* (*Suc ?Least*) *xs*))
using *kfilter-eq-LCons-1*[*of P n xs x21 x22*] **by** (*meson LCons*)
show *?thesis*
by (*metis 1 2 LCons*)
qed
qed

lemma *kfilter-lnth-zero*:
assumes ¬*lnull*(*kfilter P n xs*)
shows *lnth* (*kfilter P n xs*) *0* = *n+* (*lleast P xs*)
using *assms*
by (*metis kfilter-eq-LCons-1 lhd-LCons-ltl lnth-0-conv-lhd*)

lemma *length-LEAST-a*:
assumes ¬*lnull*(*kfilter P n xs*)

shows *lleast P xs< llength xs*
using *assms exist-lset-lnth[of xs P] kfilter-not-lnull-conv[of P n xs]*
llength-LEAST[of xs P] **by** *blast*

lemma *kfilter-upperbound*:
assumes $i < llength(kfilter\ P\ n\ xs)$
shows $(lnth\ (kfilter\ P\ n\ xs)\ i) < n + llength\ xs$
proof (*cases lfinite* (*kfilter P n xs*))
case *True*
then show *?thesis* **using** *assms*
proof (*induct zs≡kfilter P n xs arbitrary*: *xs n i rule*: *lfinite-induct*)
case (*LNil xs*)
then show *?case*
using *gr-implies-not-zero llength-lnull* **by** *blast*
next
case (*LCons xs*)
then show *?case*
proof −
let *?Least* = (*lleast P xs*)
have *1*: $\exists\ x\ xs'.\ kfilter\ P\ n\ xs = LCons\ x\ xs'$
using *LCons.hyps*(*2*) *kfilter-ldropWhile* **by** *blast*
obtain *x xs′* **where** *2*:*kfilter P n xs = LCons x xs′*
using *1* **by** *auto*
have *3*: *x = n+?Least*
using *kfilter-ldropWhile*[*of P n xs*] **by** (*simp add*: *2*)
have *4*: $i=0 \Longrightarrow (lnth\ (kfilter\ P\ n\ xs)\ i) = x$
by (*simp add*: *2*)
have *5*: *enat n+ enat ?Least < enat n + llength xs*
using *length-LEAST-a*[*of P n xs*] *LCons.hyps*(*2*) *enat-add-mono* **by** *blast*
have *6*: $i=0 \Longrightarrow enat\ (lnth\ (kfilter\ P\ n\ xs)\ i) < enat\ n + llength\ xs$
using *3 4 5* **by** *auto*
have *7*: *xs′ = kfilter P* (*Suc* (*n+?Least*)) (*ldrop* (*Suc ?Least*) *xs*)
using *kfilter-ldropWhile*[*of P n xs*] *2 kfilter-eq-LCons-1* **by** *blast*
have *8*: $i>0 \Longrightarrow enat\ (lnth\ (kfilter\ P\ n\ xs)\ i) = enat\ (lnth\ xs'\ (i-1))$
by (*simp add*: *2 lnth-LCons′*)
have *9*: $i>0 \wedge lnull\ xs' \Longrightarrow enat\ (lnth\ (kfilter\ P\ n\ xs)\ i) < enat\ n + llength\ xs$
by (*metis 2 LCons.prems*(*1*) *One-nat-def enat-ord-simps*(*2*) *llength-LCons llength-LNil*
llist.collapse(*1*) *not-less-eq one-eSuc one-enat-def*)
have *10*: $i>0 \wedge \neg lnull\ xs' \Longrightarrow (i-1) < llength\ xs'$
using *2 LCons.prems*(*1*) *Suc-ile-eq* **by** *fastforce*
have *11*: $i>0 \wedge \neg lnull\ xs' \Longrightarrow$
$enat\ (lnth\ xs'\ (i-1)) < enat\ (Suc\ (n+?Least)) + llength\ (ldrop\ (Suc\ ?Least)\ xs)$
by (*metis* (*no-types*, *lifting*) *10 2 7 LCons.hyps*(*3*) *ltl-simps*(*2*))
have *111*: *lappend* (*ltake* (*Suc* (*?Least*)) *xs*) (*ldrop* (*Suc* (*?Least*)) *xs*) = *xs*
using *lappend-ltake-ldrop* **by** *blast*
have *112*: *enat* (*Suc* (*n+?Least*)) = *n* + (*Suc ?Least*)
by *auto*
have *113*: *Suc ?Least* ≤ (*llength xs*)
using *5 Suc-ile-eq enat-add-mono* **by** *blast*
have *114*: *llength* (*ltake* (*Suc* (*?Least*)) *xs*) = *Suc ?Least*

```
      using llength-ltake[of (Suc (?Least)) xs]  113 by linarith
    have 12: enat (Suc (n+?Least)) + llength (ldrop (Suc ?Least) xs) ≤ enat n + llength xs
      using llength-lappend[of (ltake (Suc (?Least)) xs) (ldrop (Suc (?Least)) xs)]
      by (metis (no-types, lifting) 111 112 114 eq-refl group-cancel.add1 plus-enat-simps(1))
    have 14: i>0 ∧ ¬lnull xs' ⟹ enat (lnth (kfilter P n xs) i) < enat n + llength xs
      using 11 12 8 by auto
    have 15: i>0 ⟹ enat (lnth (kfilter P n xs) i) < enat n + llength xs
      using 14 9 dual-order.strict-implies-order by blast
    show ?thesis
      using 15 6 by blast
  qed
 qed
next
case False
then show ?thesis
by (metis enat-ord-simps(4) iadd-le-enat-iff kfilter-llength leD leI llength-eq-enat-lfiniteD
    llength-eq-infty-conv-lfinite llength-lfilter-ile)
qed

lemma kfilter-lowerbound:
assumes  i < llength(kfilter P n xs)
 shows  n ≤(lnth (kfilter P n xs) i)
using assms
proof (induct i arbitrary: xs n)
case 0
then show ?case
using kfilter-ldropWhile zero-enat-def by fastforce
next
case (Suc i)
then show ?case
 proof −
  let ?Least = lleast P xs
  have 7: ∃ x xs'. (kfilter P n xs) = LCons x xs'
    using Suc.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull by blast
  obtain x xs' where 8: (kfilter P n xs) = LCons x xs'
    using 7 by auto
  have 9: lnth (kfilter P n xs) (Suc i) = lnth xs' i
    by (simp add: 8)
  have 10: xs' = kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)
    using kfilter-ldropWhile[of P n xs] 8 kfilter-eq-LCons-1 by blast
  have 11: enat i < llength (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))
    by (metis 10 8 Extended-Nat.eSuc-mono Suc.prems(1) eSuc-enat llength-LCons)
  have 12: lnull xs'  ⟹ n ≤ lnth (kfilter P n xs) (Suc i)
    using 10 11 gr-implies-not-zero llength-eq-0 by blast
  have 13: ¬lnull xs' ⟹ (Suc (n+?Least)) ≤ lnth xs' i
    using 10 11 Suc.hyps by blast
  have 14: ¬ lnull xs'  ⟹ n ≤ lnth (kfilter P n xs) (Suc i)
    using 13 9 by linarith
  show ?thesis using 12 14 by blast
 qed
```

qed

lemma *kfilter-mono*:
assumes $(Suc\ i) < llength(kfilter\ P\ n\ xs)$
shows $(lnth\ (kfilter\ P\ n\ xs)\ i) < (lnth\ (kfilter\ P\ n\ xs)\ (Suc\ i))$
using *assms*
proof (*induct i arbitrary*: *xs n*)
case *0*
then show *?case*
proof −
let *?Least* = *lleast P xs*
have *1*: ∃ *x xs′*. *kfilter P n xs* = *LCons x xs′*
using *0.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull* **by** *blast*
obtain *x xs′* **where** *2*:*kfilter P n xs* = *LCons x xs′*
using *1* **by** *auto*
have *3*: *x* = *n*+*?Least*
using *kfilter-ldropWhile*[*of P n xs*] **by** (*simp add*: *2*)
have *4*: *lnth* (*kfilter P n xs*) *0* = *n*+*?Least*
by (*simp add*: *2 3*)
have *5*: *lnth* (*kfilter P n xs*) (*Suc 0*) = *lnth xs′ 0*
by (*simp add*: *2*)
have *6*: *xs′* = *kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)
using *kfilter-ldropWhile*[*of P n xs*] *2 kfilter-eq-LCons-1* **by** *blast*
have *7*: *n*+*?Least* < *lnth xs′ 0*
by (*metis 0.prems 2 6 Extended-Nat.eSuc-mono One-nat-def Suc-le-lessD kfilter-lowerbound*
llength-LCons one-eSuc one-enat-def zero-enat-def)
show *?thesis* **by** (*simp add*: *4 5 7*)
qed
next
case (*Suc i*)
then show *?case*
proof −
let *?Least* = *lleast P xs*
have *8*: ∃ *x xs′*. *kfilter P n xs* = *LCons x xs′*
using *Suc.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull* **by** *blast*
obtain *x xs′* **where** *9*:*kfilter P n xs* = *LCons x xs′*
using *8* **by** *auto*
have *10*: *xs′* = *kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)
using *kfilter-ldropWhile*[*of P n xs*] *9 kfilter-eq-LCons-1* **by** *blast*
have *11*: *lnth* (*kfilter P n xs*) (*Suc* (*Suc i*)) = *lnth xs′* (*Suc i*)
by (*simp add*: *9*)
have *12*: *lnth* (*kfilter P n xs*) (*Suc i*) < *lnth xs′* (*Suc i*)
by (*metis* (*no-types*, *lifting*) *10 9 Extended-Nat.eSuc-mono Suc*(*1*) *Suc*(*2*) *eSuc-enat*
llength-LCons lnth-Suc-LCons)
show *?thesis* **by** (*simp add*: *11 12*)
qed
qed

lemma *lfilter-kfilter*:
assumes *i* < *llength*(*kfilter P n xs*)

shows (*lnth xs* ((*lnth* (*kfilter P n xs*) *i*)−*n*)) = (*lnth* (*lfilter P xs*) *i*)
using *assms*
proof (*induct i arbitrary*: *xs n*)
case *0*
then show *?case*
proof −
let *?Least* = *lleast P xs*
have *1*: *lnth* (*kfilter P n xs*) *0* = *n* + *?Least*
using *0.prems kfilter-ldropWhile zero-enat-def* **by** *fastforce*
have *2*: (*lnth xs* ((*lnth* (*kfilter P n xs*) *0*)−*n*)) =
(*lnth xs ?Least*)
by (*simp add*: *1*)
have *3*: (*lnth* (*lfilter P xs*) *0*) = *lhd* (*ldropWhile* (*Not* ∘ *P*) *xs*)
by (*metis* (*full-types*) *0.prems kfilter-llength lhd-conv-lnth lhd-lfilter llength-eq-0 not-iless0*)
have *4*: ∃ *n*<*llength xs*. *P* (*lnth xs n*)
by (*metis 0.prems dual-order.irrefl in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter*
zero-enat-def)
have *5*: (*ldropWhile* (*Not* ∘ *P*) *xs*) = *ldropn ?Least xs*
using *ldropWhile-LEAST-not*[*of xs P*] *4* **unfolding** *lleast-def* **by** *blast*
have *6*: *lhd* (*ldropn ?Least xs*) = (*lnth xs ?Least*)
by (*simp add*: *4 lhd-ldropn llength-LEAST*)
show *?thesis* **using** *2 3 5 6* **by** *auto*
qed
next
case (*Suc i*)
then show *?case*
proof −
let *?Least*= *lleast P xs*
have *7*: ∃ *x xs*′. (*kfilter P n xs*) = *LCons x xs*′
using *Suc.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull* **by** *blast*
obtain *x xs*′ **where** *8*: (*kfilter P n xs*) = *LCons x xs*′
using *7* **by** *auto*
have *9*: *lnth* (*kfilter P n xs*) (*Suc i*) = *lnth xs*′ *i*
by (*simp add*: *8*)
have *10*: *xs*′ = *kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)
using *kfilter-ldropWhile*[*of P n xs*] **by** (*simp add*: *8 kfilter-eq-LCons-1*)
have *11*: *enat i* < *llength* (*kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*))
by (*metis 10 8 Extended-Nat.eSuc-mono Suc.prems*(*1*) *eSuc-enat llength-LCons*)
have *12*: *lnull xs*′ ⟹
lnth xs (*lnth* (*kfilter P n xs*) (*Suc i*) − *n*) = *lnth* (*lfilter P xs*) (*Suc i*)
by (*metis 10 11 llength-LNil llist.collapse*(*1*) *not-less-zero*)
have *13*: ¬*lnull xs*′ ⟹
lnth (*ldrop* (*Suc ?Least*) *xs*) (*lnth xs*′ *i* − (*Suc* (*n*+*?Least*))) =
lnth (*lfilter P* (*ldrop* (*Suc ?Least*) *xs*)) *i*
by (*metis* (*no-types*, *lifting*) *10 11 Suc.hyps*)
have *14*: (*lnth* (*lfilter P xs*) (*Suc i*)) = (*lnth* (*ltl*(*lfilter P xs*)) *i*)
by (*metis 8 kfilter-not-lnull-conv llist.disc*(*2*) *lnth-ltl lnull-lfilter*)
have *15*: (*ltl*(*lfilter P xs*)) = *lfilter P* (*ltl* (*ldropWhile* (*Not* ∘ *P*) *xs*))
using *ltl-lfilter* **by** *blast*
have *16*: (*ltl* (*ldropWhile* (*Not* ∘ *P*) *xs*)) =

```
        (ltl (ldropn (LEAST n. n < llength xs ∧ (Not ∘(Not∘ P)) (lnth xs n)  ) xs))
    using ldropWhile-LEAST[of xs Not ∘ P]
    by (metis (no-types, lifting) 8 comp-apply in-lset-conv-lnth kfilter-lnull-conv llist.disc(2))
  have 17:  (ltl (ldropn (LEAST n. n < llength xs ∧ (Not ∘(Not∘ P)) (lnth xs n)  ) xs)) =
        (ltl (ldropn ?Least xs))
    unfolding lleast-def by auto
  have 18: (ltl (ldropn ?Least xs)) =
        ldropn (Suc ?Least) xs
    by (simp add: ldropn-ltl ltl-ldropn)
  have 19: ldropn (Suc ?Least) xs =
        ldrop (Suc ?Least) xs
    by (simp add: ldrop-enat)
  have 20: lnth (lfilter P xs) (Suc i) = lnth (lfilter P (ldrop (Suc ?Least) xs)) i
    using 14 15 16 18 19 unfolding lleast-def by auto
  have 21: lnth xs (lnth (kfilter P n xs) (Suc i) − n) = lnth xs (lnth xs' i −n)
    by (simp add: 10 9)
  have 22: n≤ lnth xs' i
    using 9 Suc.prems(1) kfilter-lowerbound by fastforce
  have 23:  (Suc (n+?Least)) ≤ lnth (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)) i
    using  kfilter-lowerbound[of i P Suc (n+?Least) (ldrop (Suc ?Least) xs)  ] 11 by force
  have 24: (Suc ?Least) > llength (ltake ?Least xs)
    by (simp add: min.strict-coboundedI1)
  have 25: (ldrop (Suc ?Least) (lappend (ltake ?Least xs) (ldrop ?Least xs))) =
         ldrop ((Suc ?Least) − llength(ltake ?Least xs) ) (ldrop ?Least xs)
    by (simp add: ldrop-lappend)
  have 26: lappend (ltake (Suc ?Least) xs) (ldrop (Suc ?Least) xs) = xs
    by (simp add: lappend-ltake-ldrop)
  have 27:  (Suc ?Least) ≤ (lnth xs' i −n)
    using 10 23 by auto
  have 271:  lnth xs' i −n − (Suc ?Least) = lnth xs' i − (Suc (n+?Least))
    by auto
  have 28: lnth (lappend (ltake (Suc ?Least) xs) (ldrop (Suc ?Least) xs)) (lnth xs' i −n) =
          lnth (ldrop (Suc ?Least) xs) (lnth xs' i − (Suc (n+?Least)))
    using lnth-lappend[of (ltake (Suc ?Least) xs) (ldrop (Suc ?Least) xs) (lnth xs' i −n)]
       27 271 11 19
    by (metis (no-types, lifting) gr-implies-not-zero kfilter-LNil ldropn-eq-LNil
       le-cases llength-lnull llength-ltake llist.disc(1) lnth-lappend2 min-def)
  have 29: lnth xs (lnth xs' i −n) =
          lnth (ldrop (Suc ?Least) xs) (lnth xs' i − (Suc (n+?Least)))
    using 28 by (metis (no-types, lifting) 26)
  have 30:  ¬lnull xs' ⟹
         lnth xs (lnth (kfilter P n xs) (Suc i) − n) = lnth (lfilter P xs) (Suc i)
    by (metis 13 20 21 29)
  show ?thesis
    using 12 30 by blast
 qed
qed

lemma in-kfilter-lset:
 shows  x ∈ lset (kfilter P n xs) ⟷ x ∈ { n+i | i. i<llength xs ∧ P (lnth xs i) }
```

```
 (is ?lhs = ?rhs)
proof
 assume ?lhs
  thus ?rhs
   proof (induct zs≡kfilter P n xs arbitrary: n xs rule:llist-set-induct)
   case (find)
   then show ?case
    proof −
      let ?Least = lleast P xs
      have 1: lhd (kfilter P n xs) = n+?Least
        using kfilter-ldropWhile[of P n xs] find by auto
      have 2: P (lnth xs ?Least) unfolding lleast-def
        by (metis (mono-tags, lifting) LeastI exist-lset-lnth find.hyps kfilter-lnull-conv)
      have 3: ?Least < llength xs
        by (metis find.hyps length-LEAST-a )
      have 4: n+?Least ∈ {n + i |i. enat i < llength xs ∧ P (lnth xs i)}
        using 2 3 by blast
      show ?thesis using 1 4 by auto
    qed
 next
 case (step y)
 then show ?case
    proof −
     let ?Least = lleast P xs
     have 5: ltl (kfilter P n xs) = kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)
       using kfilter-ldropWhile[of P n xs]
       by (metis kfilter-def ltl-simps(2) step.hyps(1))
     have 6: lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)) ⟹
             y ∈ {n + i |i. enat i < llength xs ∧ P (lnth xs i)}
       by (metis 5 gr-implies-not-zero in-lset-conv-lnth llength-eq-0 step.hyps(2))
     have 7: ¬ lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)) ⟹
              y ∈ {(Suc (n+?Least)) + i |i. enat i < llength (ldrop (Suc ?Least) xs) ∧
                                          P (lnth (ldrop (Suc ?Least) xs) i)}
       using step.hyps using 5 by blast
     have 8: (Suc (n+?Least)) = n + Suc ?Least
       by auto
     have 9: ¬ lnull(kfilter P (Suc (n + ?Least)) (ldrop (enat (Suc ?Least)) xs)) ⟹
             ∃ i. y = n + Suc (?Least) + i ∧ enat i < llength (ldrop (enat (Suc ?Least)) xs) ∧
                P (lnth (ldrop (enat (Suc ?Least)) xs) i) ⟹
              ∃ i. y = n + i ∧ enat i < llength xs ∧ P (lnth xs i)
      proof −
       assume a0: ¬ lnull(kfilter P (Suc (n + ?Least)) (ldrop (enat (Suc ?Least)) xs))
       assume a1: ∃ i. y = n + Suc (?Least) + i ∧ enat i < llength (ldrop (enat (Suc ?Least)) xs) ∧
                       P (lnth (ldrop (enat (Suc ?Least)) xs) i)
       show ∃ i. y = n + i ∧ enat i < llength xs ∧ P (lnth xs i)
       proof −
        obtain i where 10: y = n + Suc (?Least) + i ∧ enat i < llength (ldrop (enat (Suc ?Least)) xs) ∧
                             P (lnth (ldrop (enat (Suc ?Least)) xs) i)
          using a1 by auto
        have 11: Suc (?Least) + i < llength xs
```

```
        by (metis 10 add.commute ldrop-enat ldrop-ldrop leD le-less-linear lnull-ldropn
            plus-enat-simps(1))
       have 12: P (lnth xs (Suc (?Least) + i))
        by (metis 10 11 add.commute ldrop-enat lnth-ldropn)
       show ?thesis
        using 10 11 12 ab-semigroup-add-class.add-ac(1) by blast
      qed
     qed
   have 13: ¬ lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)) ⟹
          y ∈ {n + i |i. enat i < llength xs ∧ P (lnth xs i)}
    using 7 9 by auto
   show ?thesis
    using 13 6 by blast
  qed
 qed
next
 assume ?rhs
 then obtain i where 15: x = n+i ∧ enat i < llength xs ∧ P (lnth xs i)
  by blast
 thus ?lhs
  proof (induct i arbitrary: xs n)
  case 0
  then show ?case
   proof −
    have 16:  lhd (kfilter P n xs) = n+(lleast P xs)
     using kfilter-ldropWhile[of P n xs] using 0.prems
     by (metis in-lset-conv-lnth kfilter-lnull-conv lhd-LCons)
    show ?thesis
     by (metis 0.prems add.right-neutral kfilter-LCons ldropn-0 ldropn-Suc-conv-ldropn
         llist.set-intros(1))
   qed
  next
  case (Suc i)
  then show ?case
   proof −
    have 18: lnull xs ⟹ x ∈ lset (kfilter P n xs)
     using Suc(2) by auto
    have 19: ¬ lnull xs ⟹ P (lnth xs (Suc i)) = P (lnth (ltl xs) ( i))
     by (simp add: lnth-ltl)
    have 20: ¬ lnull xs ⟹ x = (Suc n) +i ∧ enat i < llength (ltl xs) ∧ P (lnth (ltl xs) ( i))
     by (metis 19 Extended-Nat.eSuc-mono Suc.prems(1) add-Suc-shift eSuc-enat lhd-LCons-ltl
         llength-LCons)
    have 21: ¬ lnull xs ⟹ lnull (kfilter P n (ltl xs)) ⟹ x ∈ lset (kfilter P n xs)
     by (metis 20 in-lset-conv-lnth kfilter-lnull-conv)
    have 22: ¬ lnull xs ⟹ ¬lnull (kfilter P n (ltl xs)) ⟹ x ∈ lset (kfilter P (Suc n) (ltl xs))
     by (metis 20 Suc.hyps)
    have 23: ¬lnull xs ⟹ x ∈ lset (kfilter P n xs)
     using kfilter-LCons[of P n lhd xs ltl xs]
          in-lset-ltlD lhd-LCons-ltl[of (kfilter P n (ltl xs))]
     using 21 22 by fastforce
```

```
      show ?thesis
        using 18 23 by blast
    qed
  qed
qed

lemma kfilter-lset:
 shows  lset (kfilter P n xs) = { n+i | i. i<llength xs ∧ P (lnth xs i) }
using  in-kfilter-lset by blast

lemma lfilter-lnth-exist:
 assumes
        i < llength (lfilter P xs)
 shows  (∃ k < llength xs. lnth (lfilter P xs) i = lnth xs k)
using assms lset-lfilter[of P xs]
by (metis (no-types, opaque-lifting) add.left-neutral diff-zero kfilter-llength kfilter-upperbound
lfilter-kfilter zero-enat-def)

lemma ldistinct-kfilter:
  ldistinct(kfilter P n xs)
proof (coinduction arbitrary: n xs)
case (ldistinct n1 xs1)
then show ?case
   proof −
    have 1: lhd (kfilter P n1 xs1) ∉ lset (ltl (kfilter P n1 xs1))
      proof −
       have f1: kfilter P n1 xs1 =
                LCons (n1 + llleast P xs1)
                    (lmap snd
                        (lfilter (P ∘ fst)
                                (lzip
                                    (ldrop (enat (Suc (llleast P xs1))) xs1)
                                    (iterates Suc (Suc (n1 + llleast P xs1))))))
         by (meson kfilter-ldropWhile ldistinct)
       then have lmap snd
                    (lfilter (P ∘ fst)
                            (lzip (ldrop (enat (Suc (llleast P xs1))) xs1)
                                (iterates Suc (Suc (n1 + llleast P xs1))))) =
              kfilter P (Suc (n1 + llleast P xs1)) (ldrop (enat (Suc (llleast P xs1))) xs1)
       using kfilter-eq-LCons-1 by blast
       then show ?thesis
       using f1 by (metis (no-types) in-lset-conv-lnth kfilter-lowerbound leD lessI lhd-LCons ltl-simps(2))
      qed
    have 2: ((∃ n xs. ltl (kfilter P n1 xs1) = kfilter P n xs) ∨ ldistinct (ltl (kfilter P n1 xs1)))
      by (metis kfilter-eq-LCons-1 ldistinct llist.discI(1) llist.exhaust-sel)
    show ?thesis
      using 1 2 by auto
 qed
qed
```

lemma *kfilter-llength-ltake*:
llength(*kfilter P n* (*ltake k xs*)) ≤ *llength*(*kfilter P n xs*)
by (*simp add*: *kfilter-llength lprefix-lfilterI lprefix-llength-le*)

lemma *kfilter-ldropn-lset*:
assumes *k*< *llength xs*
shows *lset*(*kfilter P n* (*ldropn k xs*)) =
{ *n*+*i* | *i*. *i*< *llength xs* − *k* ∧ *P* (*lnth xs* (*k*+*i*)) }
using *assms*
kfilter-lset[*of P n* (*ldropn k xs*)]
by *auto*
(*metis* (*no-types*, *lifting*) *add.commute enat-min lnth-ldropn order.strict-implies-order*
plus-enat-simps(*1*),
metis (*no-types*, *lifting*) *add.commute dual-order.strict-implies-order enat-min lnth-ldropn*
plus-enat-simps(*1*))

lemma *kfilter-ldropn-lset-a*:
assumes *k*< *llength xs*
shows *lset*(*kfilter P n* (*ldropn k xs*)) =
{ *n*+(*i*−*k*) | *i*. *k*≤*i* ∧ *i*< *llength xs* ∧ *P* (*lnth xs i*) }
proof −
have *1*: ⋀*x*. *x*∈ { *n*+*i* | *i*. *i*< *llength xs* − *k* ∧ *P* (*lnth xs* (*k*+*i*)) } ⟷
x ∈ { *n*+(*i*−*k*) | *i*. *k*≤*i* ∧ *i*< *llength xs* ∧ *P* (*lnth xs i*) }
proof *auto*
show ⋀*i*. *enat i* < *llength xs* − *enat k* ⟹
P (*lnth xs* (*k* + *i*)) ⟹
∃ *ia*. *i* = *ia* − *k* ∧ *k* ≤ *ia* ∧ *enat ia* < *llength xs* ∧ *P* (*lnth xs ia*)
using *assms*
by (*metis add.commute add-diff-cancel-left*′ *enat-min le-add1 less-imp-le plus-enat-simps*(*1*))
show ⋀*i*. *k* ≤ *i* ⟹
enat i < *llength xs* ⟹
P (*lnth xs i*) ⟹
∃ *ia*. *n* + *i* − *k* = *n* + *ia* ∧ *enat ia* < *llength xs* − *enat k* ∧ *P* (*lnth xs* (*k* + *ia*))
using *assms ldropn-Suc-conv-ldropn*[*of - xs*] *ldropn-eq-LConsD*[*of - ldropn k xs*]
ldropn-ldropn[*of - - xs*]
by (*metis Nat.add-diff-assoc le-add-diff-inverse le-add-diff-inverse2 llength-ldropn*)
qed
show *?thesis* **using** *assms 1 kfilter-ldropn-lset*[*of k xs P n*] **by** *auto*
qed

lemma *kfilter-ldropn-lset-b*:
assumes *k*< *llength xs*
shows *lset*(*kfilter P n* (*ldropn k xs*)) =
{ *n*+*i* | *i*. *i*< *llength xs* −*k* ∧ *P* (*lnth xs* (*i*+*k*)) }
proof −
have *1*: ⋀*x*. *x*∈{ *n*+*i* | *i*. *i*< *llength xs* − *k* ∧ *P* (*lnth xs* (*k*+*i*)) } ⟷
x∈ { *n*+*i* | *i*. *i*< *llength xs* −*k* ∧ *P* (*lnth xs* (*i*+*k*)) }
by (*auto simp add*: *add.commute*)
show *?thesis* **using** *assms 1 kfilter-ldropn-lset*[*of k xs P n*] **by** *auto*

qed

lemma *kfilter-llength-n-zero*:
shows *llength*(*kfilter P n xs*) = *llength*(*kfilter P 0 xs*)
by (*simp add*: *kfilter-llength*)

lemma *kfilter-lnth-n-zero-a*:
assumes $k < llength\ (kfilter\ P\ n\ xs)$
shows $n \leq (lnth\ (kfilter\ P\ n\ xs)\ k)$
using *assms* **by** (*simp add*: *kfilter-lnull-conv kfilter-lowerbound*)

lemma *kfilter-lnth-n-zero*:
assumes $k < llength\ (kfilter\ P\ n\ xs)$
shows $(lnth\ (kfilter\ P\ n\ xs)\ k) - n = (lnth\ (kfilter\ P\ 0\ xs)\ k)$
using *assms*
proof (*induct k arbitrary*: *xs n*)
case *0*
then show *?case* **by** (*cases* (*kfilter P n xs*))
(*simp*,
metis add.left-neutral add-left-cancel kfilter-ldropWhile kfilter-lnull-conv kfilter-lowerbound
le-add-diff-inverse llength-eq-0 lnth-0 not-gr-zero zero-enat-def)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: $lnull(kfilter\ P\ n\ xs) \Longrightarrow lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) - n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
using *Suc.prems gr-implies-not-zero llength-lnull* **by** *blast*
have *2*: $\neg lnull(kfilter\ P\ n\ xs) \Longrightarrow lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) - n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
proof −
assume *a0*: $\neg lnull(kfilter\ P\ n\ xs)$
show $lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) - n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
proof −
let *?Least* = *lleast P xs*
have *3*: $\exists\ x\ xs'.(kfilter\ P\ n\ xs) = LCons\ x\ xs'$
using *a0 kfilter-ldropWhile* **by** *blast*
obtain *x xs'* **where** *4*: $(kfilter\ P\ n\ xs) = LCons\ x\ xs'$
using *3* **by** *auto*
have *5*: $x = n + ?Least$
using *kfilter-ldropWhile*[*of P n xs*]
by (*simp add*: *4*)
have *6*: $\neg lnull(kfilter\ P\ n\ xs) \longleftrightarrow \neg lnull(kfilter\ P\ 0\ xs)$
by (*simp add*: *kfilter-not-lnull-conv*)
have *7*: $\exists\ y\ ys'.(kfilter\ P\ 0\ xs) = LCons\ y\ ys'$
using *6 a0 kfilter-ldropWhile* **by** *blast*
obtain *y ys'* **where** *8*: $(kfilter\ P\ 0\ xs) = LCons\ y\ ys'$
using *7* **by** *auto*
have *9*: $y = ?Least$
using *kfilter-ldropWhile*[*of P 0 xs*] **by** (*simp add*: *8*)
have *10*: $x - n = y$
using *5 9 diff-add-inverse* **by** *blast*

```
      have 11: xs' = kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)
        using kfilter-ldropWhile[of P n xs]
        by (metis (no-types, lifting) 4 a0 kfilter-def ltl-simps(2))
      have 12: ys' = kfilter P (Suc (0+?Least)) (ldrop (Suc ?Least) xs)
        using kfilter-ldropWhile[of P 0 xs]
        by (metis (no-types, lifting) 6 8 a0 kfilter-def ltl-simps(2))
      have 13: lnth (kfilter P n xs) (Suc k) - n  = lnth xs' k - n
        by (simp add: 4)
      have 14: lnth (kfilter P 0 xs) (Suc k)   = lnth ys' k
        by (simp add: 8)
      have 15: ¬(∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⟹
                 lnth (kfilter P n xs) (Suc k) -n = lnth (kfilter P 0 xs) (Suc k)
        using 8 Suc by auto
        (metis (full-types) gr-implies-not-zero kfilter-eq-conv kfilter-not-lnull-conv
         ldropn-Suc-LCons ldropn-Suc-conv-ldropn ldropn-eq-LConsD llength-LNil llist.disc(2))
      have 16: enat k < llength (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))
        by (metis (no-types, lifting) 12 8 Extended-Nat.eSuc-mono Suc.prems eSuc-enat
            kfilter-llength llength-LCons)
      have 17: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⟹
             lnth xs' k  - (Suc (n+?Least)) = lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k
        using Suc.hyps using 11 16 by blast
      have 18: enat k < llength (kfilter P (Suc (?Least)) (ldrop (Suc ?Least) xs))
        by (metis 16 kfilter-llength)
      have 19: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⟹
               lnth ys' k - (Suc (0+?Least)) =
               lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k
        using Suc.hyps by (simp add: 12 18)
      have 20: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⟹
                 lnth (kfilter P n xs) (Suc k) -n = lnth xs' k -n
        using 13 by blast
      have 21: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⟹
                 lnth xs' k -n = lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k + (Suc (?Least))
        using 11 16 17 kfilter-lowerbound by fastforce
      have 22: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⟹
                 lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k + (Suc (?Least)) = lnth ys' k
        using 12 18 19 kfilter-lowerbound by fastforce
      have 23: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⟹
                 lnth (kfilter P n xs) (Suc k) -n = lnth (kfilter P 0 xs) (Suc k)
        using 14 20 21 22 by linarith
      show ?thesis
        using 15 23 by blast
    qed
  qed
  show ?thesis
  using 1 2 by blast
 qed
qed

lemma kfilter-n-zero:
 shows  (kfilter P n xs) = lmap (λi. i+n) (kfilter P 0 xs)
```

proof −
have *1*: *llength*(*kfilter P n xs*) = *llength* (*lmap* (λ*i*. *i*+*n*) (*kfilter P 0 xs*))
by (*simp add*: *kfilter-llength*)
have *2*: ⋀*k*. *k* < *llength*(*kfilter P n xs*) ⟶
lnth (*kfilter P n xs*) *k* = *lnth* (*lmap* (λ*i*. *i*+*n*) (*kfilter P 0 xs*)) *k*
using *kfilter-lnth-n-zero kfilter-lnth-n-zero-a*
by (*metis 1 le-add-diff-inverse2 llength-lmap lnth-lmap*)
show *?thesis*
by (*simp add*: *1 2 llist-eq-lnth-eq*)
qed

lemma *kfilter-n-zero-a*:
shows (*kfilter P 0 xs*) = *lmap* (λ*i*. *i*−*n*) (*kfilter P n xs*)
proof −
have *1*: *llength*(*kfilter P 0 xs*) = *llength* (*lmap* (λ*i*. *i*−*n*) (*kfilter P n xs*))
by (*simp add*: *kfilter-llength*)
have *2*: ⋀*k*. *k* < *llength*(*kfilter P 0 xs*) ⟶
lnth (*kfilter P 0 xs*) *k* = *lnth* (*lmap* (λ*i*. *i*−*n*) (*kfilter P n xs*)) *k*
by (*simp add*: *kfilter-llength kfilter-lnth-n-zero*)
show *?thesis*
using *1 2 llist-eq-lnth-eq* **by** *blast*
qed

lemma *kfilter-holds*:
assumes *y* ∈ *lset*(*kfilter P n xs*)
shows *P* (*lnth xs* (*y*−*n*))
using *assms in-kfilter-lset*[*of y P n xs*] *kfilter-lnull-conv*[*of P n xs*]
using *lset-lnull* **by** *fastforce*

lemma *kfilter-holds-not*:
assumes *y* ∈ ({*i*+*n*| *i*. *i* < *llength xs*} − (*lset* (*kfilter P n xs*)))
shows ¬ *P* (*lnth xs* (*y*−*n*))
using *assms kfilter-lset*[*of P n xs*] *kfilter-lnull-conv*[*of P n xs*]
by *auto*

lemma *kfilter-holds-a*:
assumes *i* < *llength xs*
(*i*+*n*) ∈ *lset*(*kfilter P n xs*)
shows *P* (*lnth xs i*)
using *assms kfilter-holds*[*of i*+*n P n xs*] **by** *simp*

lemma *kfilter-holds-not-a*:
assumes *i* < *llength xs*
P (*lnth xs i*)
shows (*i*+*n*) ∈ *lset*(*kfilter P n xs*)
using *assms*
by (*simp add*: *in-kfilter-lset kfilter-lnull-conv*)

lemma *kfilter-holds-b*:
assumes *i* < *llength xs*

shows $(i+n) \in lset(kfilter\ P\ n\ xs) = \quad P\ (lnth\ xs\ i)$
using *assms*
by (*meson kfilter-holds-a kfilter-holds-not-a*)

lemma *kfilter-holds-c*:
assumes $n \leq i$
$i - n < llength\ xs$
shows $i \in lset(kfilter\ P\ n\ xs) = P\ (lnth\ xs\ (i-n))$
using *assms*
by (*metis diff-add idiff-enat-enat kfilter-holds kfilter-holds-not-a*)

lemma *kfilter-holds-not-b*:
assumes $n \leq i$
$i - n < llength\ xs$
shows $i \notin lset(kfilter\ P\ n\ xs) = (\neg\ P\ (lnth\ xs\ (i-n)))$
using *assms* **by** (*simp add*: *kfilter-holds-c*)

lemma *kfilter-disjoint-lset-coset*:
shows $(\{i+n|\ i.\ i < llength\ xs\} - (lset\ (kfilter\ P\ n\ xs))) \cap lset\ (kfilter\ P\ n\ xs) = \{\}$
by *blast*

lemma *lidx-kfilter-expand*:
assumes $(Suc\ na) < llength(kfilter\ P\ n\ xs)$
shows $lnth\ (kfilter\ P\ n\ xs)\ na < lnth\ (kfilter\ P\ n\ xs)\ (Suc\ na)$
using *assms kfilter-mono* **by** *force*

lemma *lidx-kfilter*:
shows $lidx\ \ (kfilter\ P\ n\ xs)$
unfolding *lidx-def*
using *lidx-kfilter-expand* **by** *blast*

lemma *lidx-kfilter-gr-eq*:
assumes
$k \leq j$
$j < llength(kfilter\ P\ n\ xs)$
shows $lnth\ (kfilter\ P\ n\ xs)\ k \leq lnth\ (kfilter\ P\ n\ xs)\ j$
using *assms*
using *lidx-kfilter lidx-less-eq* **by** *blast*

lemma *lidx-kfilter-gr*:
shows $\forall\ j\ .\ k<j \wedge j < llength(kfilter\ P\ n\ xs) \longrightarrow$
$lnth\ (kfilter\ P\ n\ xs)\ k < lnth\ (kfilter\ P\ n\ xs)\ j$
using *less-imp-Suc-add lidx-kfilter lidx-less* **by** *blast*

lemma *kfilter-not-before*:
assumes $0<llength(kfilter\ P\ 0\ xs)$
$i< lnth\ (kfilter\ P\ 0\ xs)\ 0$
shows $\neg\ P\ (lnth\ xs\ i)$
proof −
have *0*: $(lnth\ (kfilter\ P\ 0\ xs)\ 0)\ < llength\ xs$

```
      by (metis assms(1) gen-llength-def kfilter-upperbound llength-code zero-enat-def)
 have 1: ¬ lnull (kfilter P 0 xs)
   using assms(1) by auto
 have 2:  i ∈ lset (kfilter P 0 xs) ⟹
          ¬ lnull (kfilter P 0 xs) ⟹
          i< lnth (kfilter P 0 xs) 0 ⟹
          False
   proof (induct zs≡(kfilter P 0 xs) arbitrary: xs rule: lset-induct)
   case (find xs)
   then show ?case by (metis less-not-refl3 lhd-LCons lnth-0-conv-lhd)
   next
   case (step x' xs)
   then show ?case
   proof −
   have ∀ ns n. ∃ na. ((n::nat) ∉ lset ns ∨ lnth ns na = n) ∧ (n ∉ lset ns ∨ enat na < llength ns)
   by (meson in-lset-conv-lnth)
   then show ?thesis using step
   by (metis (no-types) leD less-nat-zero-code lidx-kfilter-gr-eq llist.set-intros(2) not-le-imp-less)
   qed
   qed
 have 3:  i ∉ lset (kfilter P 0 xs) ∧ i < llength xs ⟶ ¬ P (lnth xs i)
  by (simp add: 1 in-kfilter-lset)
 have 4:  i < llength xs
   using 0 assms(2) dual-order.strict-trans enat-ord-simps(2) by blast
 show ?thesis
 using 1 2 3 4 assms(2) by blast
qed

lemma kfilter-n-not-before:
 assumes 0 < llength (kfilter P n (ldropn n xs) )
         n < llength xs
         n ≤ i
         i< lnth (kfilter P n (ldropn n xs) ) 0
  shows  ¬ P (lnth xs i)
proof −
 have 00: ¬ lnull (kfilter P n (ldrop n xs))
     by (metis assms(1) ldrop-enat less-numeral-extra(3) llength-LNil llist.collapse(1))
 have 0: lnth (kfilter P n (ldrop n xs) ) 0 < llength xs
   using assms kfilter-upperbound[of 0 P n (ldropn n xs)]
   by (metis lappend-ltake-enat-ldropn ldrop-enat llength-lappend llength-ltake min.strict-order-iff
       zero-enat-def)
 have 1: i ∈ lset (kfilter P n (ldropn n xs)) ⟹
          ¬ lnull (kfilter P n (ldropn n xs)) ⟹
          i< lnth (kfilter P n (ldropn n xs) ) 0 ⟹
          n < llength xs  ⟹
          n ≤ i⟹
          False
   proof (induct zs≡(kfilter P n (ldropn n xs)) arbitrary: n xs rule: lset-induct)
   case (find xs)
   then show ?case
```

by (*metis lnth-0 nat-less-le*)
next
case (*step x′ xs*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *in-lset-conv-lnth kfilter-eq-LCons-1 kfilter-lowerbound leD less-SucI lnth-0*)
qed
have *2*: $i \notin lset\ (kfilter\ P\ n\ (ldropn\ n\ xs)) \wedge n \leq i \wedge i < llength\ xs \longrightarrow \neg\ P\ (lnth\ xs\ i)$
using *assms kfilter-holds-not-a*[*of i*] *00*
by (*simp add*: *kfilter-ldropn-lset-a ldrop-enat*)
have *3*: $n \leq i \wedge i < llength\ xs$
using *assms 00 kfilter-ldropWhile*[*of P n ldropn n xs*]
by (*metis 0 enat-ord-simps*(*2*) *ldrop-enat less-trans*)
show *?thesis*
using *1 2 3 assms*(*1*) *assms*(*2*) *assms*(*4*) *kfilter-not-lnull-conv* **by** *auto*
qed

lemma *kfilter-not-after*:
assumes *0*<*llength*(*kfilter P 0 xs*)
$lnth\ (kfilter\ P\ 0\ xs)\ (k-1) < i$
$llength\ (kfilter\ P\ 0\ xs) = (enat\ k)$
$i < llength\ xs$
shows $\neg\ P\ (lnth\ xs\ i)$
proof −
have *0*: ¬ *lnull* (*kfilter P 0 xs*)
using *assms*(*1*) **by** *auto*
have *01*: *0*<*k*
using *0 assms*(*3*) *gr0I zero-enat-def* **by** *fastforce*
have *02*: $i \notin lset\ (kfilter\ P\ 0\ xs)$
by (*metis 01 One-nat-def Suc-pred assms*(*2*) *assms*(*3*) *diff-less enat-ord-simps*(*2*) *in-lset-conv-lnth leD less-Suc-eq-le lidx-kfilter-gr-eq zero-less-one*)
from *0 01 02* **show** *?thesis*
by (*metis add.right-neutral assms*(*4*) *kfilter-holds-not-a*)
qed

lemma *kfilter-n-not-after*:
assumes $0 < llength\ (kfilter\ P\ n\ (ldropn\ n\ xs))$
$n < llength\ xs$
$lnth\ (kfilter\ P\ n\ (ldropn\ n\ xs)\ )\ (k-1) < i$
$llength\ (kfilter\ P\ n\ (ldropn\ n\ xs)\ ) = (enat\ k)$
$i < llength\ xs$
shows $\neg\ P\ (lnth\ xs\ i)$
proof −
have *0*: ¬ *lnull* (*kfilter P n* (*ldropn n xs*))
using *assms*(*1*) **by** *auto*
have *1*: *0*<*k*
using *assms*(*1*) *assms*(*4*) **by** (*simp add*: *zero-enat-def*)
have *2*: $i \notin lset(kfilter\ P\ n\ (ldropn\ n\ xs))$
using *assms*

```
    by (metis 1 Suc-diff-1 enat-ord-simps(2) in-lset-conv-lnth leD less-Suc-eq-le
        lidx-kfilter-gr-eq order-refl)
 from  0 1 2 show ?thesis
using assms kfilter-ldropn-lset-a[of n xs P n] kfilter-lowerbound[of - P n (ldropn n xs)]
  by auto (metis One-nat-def diff-less le-trans less-imp-le zero-less-one)
qed

lemma kfilter-not-between:
 assumes  lnth (kfilter P 0 xs) (k) < i
          i<lnth (kfilter P 0 xs) (Suc k)
          (Suc k) < llength (kfilter P 0 xs)
  shows  ¬ P (lnth xs i)
proof −
 have 0: ∃ x ∈ lset xs. P x
    using assms(3) gr-implies-not-zero kfilter-not-lnull-conv by fastforce
 have 1: ¬ lnull (kfilter P 0 xs)
   by (simp add: 0 kfilter-not-lnull-conv)
 have 2: lnth (kfilter P 0 xs) (Suc k) ≤ llength xs
   using kfilter-upperbound[of Suc k P 0 xs]
   using 1 assms(3) zero-enat-def by auto
 have 3: i ∉ lset(kfilter P 0 xs)
  using assms
  by (metis Suc-ile-eq dual-order.strict-implies-order in-lset-conv-lnth leD lidx-kfilter-gr-eq
      not-less-eq-eq)
 from 1 2 3 show ?thesis
 by (metis add.right-neutral assms(2) enat-ord-simps(2) kfilter-holds-not-a less-le-trans)
qed

lemma lfilter-kfilter-ltake-lidx-a:
 assumes  k < llength(lfilter P xs)
 shows   lidx (ltake k (kfilter P n xs))
unfolding lidx-def
using assms
by (metis Suc-ile-eq kfilter-mono less-imp-le  llength-ltake lnth-ltake min.strict-boundedE)

lemma lfilter-kfilter-ldropn-lidx-a:
 assumes  k < llength(lfilter P xs)
 shows   lidx (ldropn k (kfilter P n xs))
using assms unfolding lidx-def
  proof auto
   fix na
   assume a0: enat k < llength (lfilter P xs)
   assume a1: enat (Suc na) < llength (kfilter P n xs) − enat k
   show lnth (ldropn k (kfilter P n xs)) na < lnth (ldropn k (kfilter P n xs)) (Suc na)
   proof −
    have 1: enat (k + (Suc na)) < llength (kfilter P n xs)
      proof −
       have Suc na + k = k + Suc na
         by presburger
       then show ?thesis
```

```
      by (metis (no-types) a1 ldropn-eq-LNil ldropn-ldropn leD leI llength-ldropn)
    qed
   have 2 : enat (k + na) < llength (kfilter P n xs)
    by (metis 1 Suc-ile-eq add-Suc-right less-imp-le)
   have 3: lnth (kfilter P n xs) (k + (na)) <  lnth (kfilter P n xs) (k + (Suc na))
    using 1 kfilter-mono by auto
   show ?thesis by (metis 1 2 3 add.commute lnth-ldropn)
  qed
qed

lemma lfilter-kfilter-ldropn-lidx-b:
 assumes  k < llength(lfilter P xs)
 shows   lidx (kfilter P (lnth (kfilter P n xs) k) (ldropn (lnth (kfilter P n xs) k) xs))
using assms using lidx-kfilter by blast

lemma ltake-lset:
 assumes k < llength xs
 shows    lset (ltake k xs) = {(lnth xs i) | i. i < k}
using assms
by (auto simp add: in-lset-conv-lnth lnth-ltake)
   (blast, meson less-trans lnth-ltake)

lemma ldropn-lset:
 assumes  k < llength xs
 shows  lset (ldropn k xs) = {(lnth xs i) | i. k ≤ i ∧ i < llength xs}
using assms
proof (auto simp add: in-lset-conv-lnth )
  fix n :: nat
  assume a1: enat n < llength xs − enat k
  assume enat k < llength xs
  then have enat k ≤ llength xs
    by (meson dual-order.strict-implies-order)
  then have enat n + enat k < llength xs
    using a1 by (meson enat-min)
  then show ∃ na. lnth (ldropn k xs) n = lnth xs na ∧ k ≤ na ∧ enat na < llength xs
    using a1 by (metis ldropn-ldropn le-add2 lhd-ldropn llength-ldropn plus-enat-simps(1))
next
  fix i :: nat
  assume a1: enat i < llength xs
  assume a2: k ≤ i
  have 1:  enat (i−k) < llength xs −enat k
    by (metis a1 a2 diff-add ldropn-Suc-conv-ldropn ldropn-ldropn llength-ldropn llist.disc(2)
        lnull-ldropn not-le-imp-less)
  have 2: lnth (ldropn k xs) (i−k) = lnth xs i
    by (simp add: a1 a2)
  show ∃ n. enat n < llength xs − enat k ∧ lnth (ldropn k xs) n = lnth xs i
    using 1 2 by blast
qed

lemma lfilter-kfilter-ltake-lset-eq:
```

assumes
$k < llength(lfilter\ P\ xs)$
shows $lset\ (ltake\ k\ (kfilter\ P\ 0\ xs)) =$
$lset\ (kfilter\ P\ 0\ (ltake\ ((lnth\ (kfilter\ P\ 0\ xs)\ k))\ xs))$
proof −
have *1*: $(lnth\ (kfilter\ P\ 0\ xs)\ k) < llength\ xs$
using *kfilter-llength kfilter-upperbound*
by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)
have *2*: $\exists\ x \in lset\ xs\ .\ P\ x$
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have *3*: $\{i.\ i< llength(ltake\ ((lnth\ (kfilter\ P\ 0\ xs)\ k))\ xs) \wedge$
$P\ (lnth\ (ltake\ ((lnth\ (kfilter\ P\ 0\ xs)\ k))\ xs)\ i)\} =$
$\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge P\ (lnth\ xs\ i)\}$
by (*auto simp add*: *lnth-ltake 1 order-less-subst2*)
have *5*: $\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge P\ (lnth\ xs\ i)\} =$
$\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge i \in lset(kfilter\ P\ 0\ xs)\}$
by (*auto simp add*: *1 kfilter-holds-c order-less-subst2*)
have *6*: $\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge i \in lset(kfilter\ P\ 0\ xs)\} =$
$\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge i \in \{(lnth\ (kfilter\ P\ 0\ xs)\ j) \mid j.\ j < llength(kfilter\ P\ 0\ xs)\}\}$
by (*simp add*: *lset-conv-lnth*)
have *7*: $\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge i \in \{(lnth\ (kfilter\ P\ 0\ xs)\ j) \mid j.\ j < llength(kfilter\ P\ 0\ xs)\}\} =$
$\{(lnth\ (kfilter\ P\ 0\ xs)\ j) \mid j.\ j<k\}$
by (*auto simp add*: *assms lidx-kfilter-gr kfilter-llength*)
(*metis kfilter-llength leD lidx-kfilter-gr-eq not-le-imp-less*,
metis assms enat-ord-simps(*2*) *less-trans*)
have *8*: $k < llength(kfilter\ P\ 0\ xs)$
by (*simp add*: *assms kfilter-llength*)
have *9*: $\{(lnth\ (kfilter\ P\ 0\ xs)\ j) \mid j.\ j<k\} = lset(ltake\ k\ (kfilter\ P\ 0\ xs))$
using *ltake-lset*[*of k* $(kfilter\ P\ 0\ xs)$] *8* **by** *auto*
have *10*: $lset\ (kfilter\ P\ 0\ (ltake\ ((lnth\ (kfilter\ P\ 0\ xs)\ k))\ xs)) =$
$\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge$
$P\ (lnth\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ i)\ \}$
by (*auto simp add*: *in-kfilter-lset*)
(*meson 1 enat-ord-simps*(*2*) *less-trans*)
have *11*: $(lnth\ (kfilter\ P\ 0\ xs)\ k) = llength(ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)$
by (*simp add*: *1 dual-order.strict-implies-order min-def*)
have *12*: $\{i.\ i< (lnth\ (kfilter\ P\ 0\ xs)\ k) \wedge$
$P\ (lnth\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ i)\ \} =$
$\{i.\ i< llength(ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs) \wedge$
$P\ (lnth\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ i)\ \}$
using *11* **by** *auto*
show *?thesis*
using *10 12 3 5 6 7 9* **by** *auto*
qed

lemma *lfilter-kfilter-ldropn-lset-eq*:
assumes $k < llength(lfilter\ P\ xs)$
shows $lset(kfilter\ P\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)) =$
$lset(ldropn\ \ k\ (kfilter\ P\ 0\ xs))$
proof −

have *1*: (*lnth* (*kfilter P 0 xs*) *k*) < *llength xs*
using *kfilter-llength kfilter-upperbound*
by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)
have *2*: ∃ *x* ∈ *lset xs* . *P x*
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have *10*: *lset*(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
{(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
P (*lnth xs* (*i*+(*lnth* (*kfilter P 0 xs*) *k*))) }
using *1 kfilter-ldropn-lset-b*[*of* (*lnth* (*kfilter P 0 xs*) *k*) *xs P* (*lnth* (*kfilter P 0 xs*) *k*)]
by *linarith*
have *5*: {(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
P (*lnth xs* (*i*+(*lnth* (*kfilter P 0 xs*) *k*))) } =
{(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) }
using *kfilter-holds-b*[*of - xs 0 P*] **using** *1 2*
by *auto*
(*metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less*,
metis gen-llength-def in-lset-conv-lnth kfilter-upperbound llength-code)
have *51*: {(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) } =
{(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*.
(*lnth* (*kfilter P 0 xs*) *k*) ≤ *i* + (*lnth* (*kfilter P 0 xs*) *k*) ∧
i + (*lnth* (*kfilter P 0 xs*) *k*) < *llength xs* ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) }
by *auto*
(*metis 1 enat-min less-imp-le plus-enat-simps*(*1*),
metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less)
have *52*: {(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*.
(*lnth* (*kfilter P 0 xs*) *k*) ≤ *i* + (*lnth* (*kfilter P 0 xs*) *k*) ∧
i + (*lnth* (*kfilter P 0 xs*) *k*) < *llength xs* ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) } =
{*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧ *j* ∈ *lset*(*kfilter P 0 xs*)}
by (*metis* (*no-types*, *lifting*) *add.commute le-iff-add*)
have *53*: {*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧ *j* ∈ *lset*(*kfilter P 0 xs*)} =
{*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧
j ∈ { (*lnth* (*kfilter P 0 xs*) *jj*)| *jj*. *jj*< *llength*(*kfilter P 0 xs*)}}
by (*simp add*: *lset-conv-lnth*)
have *54*: {*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧
j ∈ { (*lnth* (*kfilter P 0 xs*) *jj*)| *jj*. *jj*< *llength*(*kfilter P 0 xs*)}} =
{ (*lnth* (*kfilter P 0 xs*) *j*) | *j*. *k*≤*j* ∧ *j* < *llength* (*kfilter P 0 xs*)}
by *auto*
(*metis assms kfilter-llength lidx-kfilter-gr not-less*,
meson lidx-kfilter-gr-eq,
metis gen-llength-def kfilter-upperbound llength-code)
have *8*: *k* < *llength*(*kfilter P 0 xs*)
by (*simp add*: *assms kfilter-llength*)
have *9*: { (*lnth* (*kfilter P 0 xs*) *j*) | *j*. *k*≤*j* ∧ *j* < *llength* (*kfilter P 0 xs*)} =
lset(*ldropn k* (*kfilter P 0 xs*))
using *ldropn-lset*[*of k* (*kfilter P 0 xs*)] **using** *8* **by** *blast*
show *?thesis*

using *10 5 51 52 53 54 9* **by** *auto*
qed

lemma *kfilter-kfilter-ltake*:
assumes $k < llength(lfilter\ P\ xs)$
shows $(ltake\ k\ (kfilter\ P\ 0\ xs)) =$
$(kfilter\ P\ 0\ (ltake\ ((lnth\ (kfilter\ P\ 0\ xs)\ k))\ xs))$
using *assms*
by (*simp add*: *lfilter-kfilter-ltake-lidx-a lfilter-kfilter-ltake-lset-eq lidx-kfilter lidx-lset-eq*)

lemma *kfilter-kfilter-ldropn*:
assumes $k < llength(lfilter\ P\ xs)$
shows $(ldropn\ \ k\ (kfilter\ P\ 0\ xs)) =$
$(kfilter\ P\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs))$
using *assms*
by (*simp add*: *lfilter-kfilter-ldropn-lidx-a lfilter-kfilter-ldropn-lidx-b*
lfilter-kfilter-ldropn-lset-eq lidx-lset-eq)

lemma *kfilter-lmap-lfilter*:
shows $lmap\ (\lambda n.\ (lnth\ xs\ n))\ (kfilter\ P\ 0\ xs) = lfilter\ P\ xs$
using *lfilter-kfilter*[*of - P 0 xs*]
by (*metis* (*no-types*, *lifting*) *diff-zero kfilter-llength llength-lmap*
llist-eq-lnth-eq lnth-lmap)

lemma *lfilter-kfilter-ltake*:
assumes $k < llength(lfilter\ P\ xs)$
shows $ltake\ k\ (lfilter\ P\ xs) =$
$(lfilter\ P\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs))$
proof −
have *2*: $lmap\ (\lambda n.\ (lnth\ xs\ n))\ (kfilter\ P\ 0\ xs) = lfilter\ P\ xs$
using *kfilter-lmap-lfilter* **by** *blast*
have *3*: $ltake\ k\ (lfilter\ P\ xs) =$
$ltake\ k\ (lmap\ (\lambda n.\ (lnth\ xs\ n))\ (kfilter\ P\ 0\ xs))$
by (*simp add*: *2*)
have *4*: $ltake\ k\ (lmap\ (\lambda n.\ (lnth\ xs\ n))\ (kfilter\ P\ 0\ xs)) =$
$lmap\ (\lambda s.\ lnth\ xs\ s)\ (ltake\ k\ (kfilter\ P\ 0\ xs))$
using *ltake-lmap* **by** *blast*
have *6*: $(lfilter\ P\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)) =$
$lmap\ (\lambda s.\ (lnth\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ s))$
$(kfilter\ P\ 0\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs))$
by (*simp add*: *kfilter-lmap-lfilter*)
have *7*: $lmap\ (\lambda s.\ lnth\ xs\ s)\ (ltake\ k\ (kfilter\ P\ 0\ xs)) =$
$lmap\ (\lambda s.\ lnth\ xs\ s)\ (kfilter\ P\ 0\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs))$
by (*simp add*: *assms kfilter-kfilter-ltake*)
have *8*: $lmap\ (\lambda s.\ lnth\ xs\ s)\ (kfilter\ P\ 0\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)) =$
$lmap\ (\lambda s.\ (lnth\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ s))$
$(kfilter\ P\ 0\ (ltake\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs))$
using *kfilter-kfilter-ltake*[*of k P xs*]
by (*metis gen-llength-def kfilter-upperbound lappend-ltake-enat-ldropn ldistinct-Ex1*
ldistinct-kfilter llength-code llist.map-cong0 lnth-lappend)

show *?thesis*
  using *2 4 6 7 8* **by** *auto*
qed

lemma *kfilter-lmap-shift-ldropn*:
shows *lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
proof −
have *1*: *llength* (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) =
llength (*lmap* (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)))
  by (*simp add*: *kfilter-llength*)
have *2*: ⋀*i*. *i*< *llength* (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) ⟹
lnth (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) *i* =
lnth xs ((*lnth* (*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)+
(*lnth* (*kfilter P 0 xs*) *k*))
  by *simp*
have *3*: ⋀*i*. *i*< *llength* (*lmap* (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) ⟹
lnth (*lmap* (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*)
(*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) *i* =
lnth xs (*lnth* (*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*)
(*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)
  by *simp*
have *4*: ⋀*i*. *i*< *llength* (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) ⟹
lnth xs ((*lnth* (*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)+
(*lnth* (*kfilter P 0 xs*) *k*)) =
lnth xs (*lnth* (*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*)
(*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)
  using *kfilter-lnth-n-zero*[*of* - *P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)]
kfilter-lowerbound[*of* - *P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)]
  using *1 diff-add* **by** *fastforce*
show *?thesis*
  using *llist-eq-lnth-eq*[*of lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))]
  using *1 2 3 4* **by** *presburger*
qed

lemma *lfilter-kfilter-ldropn*:
assumes *k* < *llength*(*lfilter P xs*)
shows (*ldropn k* (*lfilter P xs*)) =
(*lfilter P* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
proof −

have *1*: $\exists\ x \in lset\ xs.\ P\ x$
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have *2*: $(lfilter\ P\ xs) = lmap\ (\lambda s.\ lnth\ xs\ s)\ (kfilter\ P\ 0\ xs)$
by (*simp add*: *kfilter-lmap-lfilter*)
have *3*: $ldrop\ k\ (lfilter\ P\ xs) =$
$ldrop\ k\ (lmap\ (\lambda s.\ lnth\ xs\ s)\ (kfilter\ P\ 0\ xs))$
by (*simp add*: *2*)
have *4*: $(ldrop\ k\ (lmap\ (\lambda s.\ lnth\ xs\ s)\ (kfilter\ P\ 0\ xs))) =$
$(lmap\ (\lambda s.\ lnth\ xs\ s)\ (ldrop\ k\ (kfilter\ P\ 0\ xs)))$
using *ldrop-lmap* **by** *blast*
have *6*: $(lfilter\ P\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)) =$
$lmap\ (\lambda s.\ lnth\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ s)$
$(kfilter\ P\ 0\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k\ )\ xs))$
by (*simp add*: *kfilter-lmap-lfilter*)
have *61*: $\bigwedge z.\ z \in lset\ ((kfilter\ P\ 0\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k\ )\ xs)))$
$\Longrightarrow (\lambda s.\ lnth\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ s)\ z =$
$(\lambda s.\ lnth\ xs\ (s + (lnth\ (kfilter\ P\ 0\ xs)\ k))\ )\ z$
using *assms in-lset-conv-lnth*[*of* - $((kfilter\ P\ 0\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k\ )\ xs)))$]
by *simp*
(*metis add.commute add.left-neutral kfilter-upperbound ldropn-eq-LNil ldropn-ldropn leD*
lnth-ldropn not-le-imp-less zero-enat-def)
have *7*: $lmap\ (\lambda s.\ lnth\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)\ s)$
$(kfilter\ P\ 0\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k\ )\ xs)) =$
$lmap\ (\lambda s.\ lnth\ xs\ (s + (lnth\ (kfilter\ P\ 0\ xs)\ k))\ )$
$(kfilter\ P\ 0\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k\ )\ xs))$
using *61* **by** *auto*
have *8*: $lmap\ (\lambda s.\ lnth\ xs\ (s + (lnth\ (kfilter\ P\ 0\ xs)\ k))\ )$
$(kfilter\ P\ 0\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k\ )\ xs)) =$
$lmap\ (\lambda s.\ lnth\ xs\ s)$
$(kfilter\ P\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs))$
by (*simp add*: *kfilter-lmap-shift-ldropn*)
have *9*: $lmap\ (\lambda s.\ lnth\ xs\ s)$
$(kfilter\ P\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ (ldropn\ (lnth\ (kfilter\ P\ 0\ xs)\ k)\ xs)) =$
$lmap\ (\lambda s.\ lnth\ xs\ s)\ (ldropn\ k\ (kfilter\ P\ 0\ xs))$
by (*simp add*: *assms kfilter-kfilter-ldropn*)
show *?thesis*
by (*metis 4 7 8 9 kfilter-lmap-lfilter ldrop-enat*)
qed

lemma *lfilter-lnth-aa*:
assumes $n < llength\ (lfilter\ P\ xs)$
shows $P\ (lnth\ (lfilter\ P\ xs)\ n)$
using *assms*
by (*meson in-lset-conv-lnth lfilter-id-conv lfilter-idem*)

lemma *exist-one-conv*:
$(\exists!\ i.\ i{<}llength\ xs \wedge P\ (lnth\ xs\ i)) \longleftrightarrow$
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j{\neq}k \longrightarrow \neg\ P\ (lnth\ xs\ j)))$
by *blast*

lemma *lfilter-llength-one-conv-a*:
assumes *llength*(*lfilter P xs*) =*1*
shows $\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j \neq k \longrightarrow \neg\ P\ (lnth\ xs\ j))$
proof −
have *1*: $P\ (lnth\ xs\ (lnth\ (kfilter\ P\ 0\ xs)\ 0))$
by (*metis assms*(*1*) *kfilter-llength kfilter-lmap-lfilter less-numeral-extra*(*1*) *lfilter-lnth-aa lnth-lmap zero-enat-def*)
have *2*: $(lnth\ (kfilter\ P\ 0\ xs)\ 0) < llength\ xs$
by (*metis assms*(*1*) *gen-llength-def kfilter-llength kfilter-upperbound less-numeral-extra*(*1*) *llength-code zero-enat-def*)
have *3*: $(\forall\ j < llength\ xs.\ j \neq (lnth\ (kfilter\ P\ 0\ xs)\ 0) \longrightarrow \neg\ P\ (lnth\ xs\ j))$
using *assms kfilter-not-after*[*of P xs*] *kfilter-not-before*[*of P xs*]
by (*metis One-nat-def diff-Suc-1 kfilter-llength linorder-neqE-nat one-enat-def zero-less-one*)
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *lfilter-llength-one-conv-c*:
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j \neq k \longrightarrow \neg\ P\ (lnth\ xs\ j))) \longleftrightarrow$
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j < k \vee k < j \longrightarrow \neg\ P\ (lnth\ xs\ j)))$
using *antisym-conv3* **by** *auto*

lemma *lfilter-llength-one-conv-d*:
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j < k \vee k < j \longrightarrow \neg\ P\ (lnth\ xs\ j))) \longleftrightarrow$
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j\ .\ j < k \longrightarrow \neg\ P\ (lnth\ xs\ j)) \wedge$
$(\forall\ j < llength\ xs.\ k < j \longrightarrow \neg\ P\ (lnth\ xs\ j)))$
by (*meson enat-ord-simps*(*2*) *less-trans*)

lemma *exist-one-lfilter-llength-one*:
assumes $(\exists!\ i.\ i < llength\ xs \wedge P\ (lnth\ xs\ i))$
shows $llength(lfilter\ P\ xs) \leq 1$
using *assms*
proof *auto*
fix *i* :: *nat*
assume *a1*: $\forall\ y\ y'.\ enat\ y < llength\ xs \wedge P\ (lnth\ xs\ y) \wedge enat\ y' < llength\ xs \wedge P\ (lnth\ xs\ y') \longrightarrow y = y'$
show $llength\ (lfilter\ P\ xs) \leq 1$
proof −
have *f1*: $\forall\ e\ ea.\ (e::enat) \leq ea \vee ea < e$
by (*meson not-le-imp-less*)
have *f3*: $llength\ (kfilter\ P\ 0\ xs) = gen\text{-}llength\ 0\ (lfilter\ P\ xs)$
by (*metis kfilter-llength llength-code*)
have *f4*: $enat\ 0 + llength\ xs = llength\ xs$
by (*metis gen-llength-def llength-code*)

have *llength xs = enat 0 + llength xs*
by (*metis* (*full-types*) *gen-llength-def llength-code*)
then have *f5*: ¬ *1 < llength* (*lfilter P xs*)
proof −
have *g1*: *Suc 0 = 0 + Suc 0*
by *auto*
have *g2*: ⋀*i. enat i < llength* (*kfilter P 0 xs*) ⟹
lnth xs (*lnth* (*kfilter P 0 xs*) *i* − *0*) = *lnth* (*lfilter P xs*) *i*
using *lfilter-kfilter*[*of - P 0 xs*] **by** *auto*
have *g3*: ⋀*k. enat k < llength* (*kfilter P 0 xs*) ⟹
lnth (*kfilter P 0 xs*) *k* − *0* = *lnth* (*kfilter P 0 xs*) *k*
using *kfilter-mono*[*of - P 0 xs*] **by** *auto*
have *g4*: ⋀*n. enat n < llength* (*lfilter P xs*) ⟹ *P* (*lnth* (*lfilter P xs*) *n*)
using *lfilter-lnth-aa*[*of - P xs*] **by** *auto*
have *g5*: *enat* (*0 + Suc 0*) = *1*
using *one-enat-def* **by** *auto*
have ¬ *1 < gen-llength 0* (*lfilter P xs*) ∨ *enat 0 < gen-llength 0* (*lfilter P xs*)
using *zero-enat-def* **by** *fastforce*
then show *?thesis*
by (*metis g1 g2 g3 g4 g5 a1 f3 f4 kfilter-upperbound ldistinct-conv-lnth ldistinct-kfilter*
llength-code n-not-Suc-n)
qed
show *?thesis*
using *f5 not-le* **by** *blast*
qed
qed

lemma *lfilter-llength-one-conv-b*:
assumes ∃ *k < llength xs. P* (*lnth xs k*) ∧
(∀ *j < llength xs. j≠k* ⟶ ¬ *P* (*lnth xs j*))
shows *llength*(*lfilter P xs*) =*1*
proof −
have *1*: *llength*(*lfilter P xs*) ≤*1*
by (*metis assms exist-one-lfilter-llength-one*)
have *2*: *0*<*llength*(*lfilter P xs*)
by (*metis assms gr-zeroI in-lset-conv-lnth llength-eq-0 lnull-lfilter*)
show *?thesis*
by (*metis 1 2 One-nat-def Suc-ile-eq dual-order.antisym one-enat-def zero-enat-def*)
qed

lemma *lfilter-llength-one-conv*:
shows (∃ *k < llength xs. P* (*lnth xs k*) ∧
(∀ *j < llength xs. j≠k* ⟶ ¬ *P* (*lnth xs j*))) ⟷
llength(*lfilter P xs*) =*1*
using *lfilter-llength-one-conv-a*[*of P xs*] *lfilter-llength-one-conv-b*[*of xs P*] **by** *blast*

lemma *lfilter-llength-one-conv-1*:
shows (∃ *k < llength xs. P* (*lnth xs k*) ∧
(∀ *j < llength xs. j*<*k* ∨ *k*<*j* ⟶ ¬ *P* (*lnth xs j*))) ⟷
llength(*lfilter P xs*) =*1*

using *lfilter-llength-one-conv*[*of xs P*] *lfilter-llength-one-conv-c*[*of xs P*]
by *blast*

lemma *lfilter-llength-one-conv-2*:
shows ($\exists$ $k <$ *llength xs*. *P* (*lnth xs k*) $\wedge$
(∀ *j* . *j*<*k* $\longrightarrow$ $\neg$ *P* (*lnth xs j*)) $\wedge$
(∀ *j* < *llength xs*. *k*<*j* $\longrightarrow$ $\neg$ *P* (*lnth xs j*))) $\longleftrightarrow$
llength(*lfilter P xs*) =*1*
using *lfilter-llength-one-conv-1*[*of xs P*] *lfilter-llength-one-conv-d*[*of xs P*]
by *blast*

lemma *lfilter-lappend-ltake*:
assumes *k* < *llength xs*
shows *lfilter P* (*ltake k xs*) = *ltake* (*llength*(*lfilter P* (*ltake k xs*))) (*lfilter P xs*)
proof −
have *1*: *lfilter P xs* =
lappend (*lfilter P* (*ltake* (*the-enat k*) *xs*)) (*lfilter P* (*ldropn* (*the-enat k*) *xs*))
by (*metis enat-ord-code*(*4*) *lappend-ltake-enat-ldropn lfilter-lappend-lfinite lfinite-ltake*)
have *2*: *ltake* (*llength*(*lfilter P* (*ltake k xs*)))
(*lappend* (*lfilter P* (*ltake* (*the-enat k*) *xs*)) (*lfilter P* (*ldropn* (*the-enat k*) *xs*))) =
lfilter P (*ltake k xs*)
by (*metis assms enat-the-enat lfilter-idem lfinite-ltake llength-eq-infty-conv-lfinite*
llength-lfilter-ile llength-ltake ltake-all ltake-lappend1 min.strict-order-iff)
show *?thesis* **using** *1 2* **by** *simp*
qed

lemma *kfilter-lappend-lfinite*:
lfinite xs ⟹
kfilter P n (*lappend xs ys*) =
lappend (*kfilter P n xs*) (*kfilter P* (*n*+ (*the-enat*(*llength xs*))) *ys*)
unfolding *kfilter-def*
proof (*induct arbitrary*: *n rule*: *lfinite.induct*)
case *lfinite-LNil*
then show *?case* **by** *simp*
next
case (*lfinite-LConsI xs x*)
then show *?case*
proof −
have *1*: (*lzip* (*lappend* (*LCons x xs*) *ys*) (*iterates Suc n*)) =
(*LCons* (*x* , *n*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*))))
by (*simp add*: *lzip.ctr*(*2*))
have *3*: *lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend* (*LCons x xs*) *ys*) (*iterates Suc n*))) =
(*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*)))))))
else lmap snd (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*)))))
using *1* **by** *auto*
have *4*: (*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*)))))))
else lmap snd (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*))))) =
(*if P x then*
(*LCons n* (*lappend* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*)))))
(*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + *the-enat* (*llength*

xs))))))))
else (*lappend* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*)))))
(*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + *the-enat* (*llength xs*))))))))
using *lfinite-LConsI.hyps*(*2*) **by** *auto*
have *5*: (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*LCons x xs*) (*iterates Suc n*)))) =
(*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*xs*) (*iterates Suc* (*Suc n*))))))
else (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*xs*) (*iterates Suc* (*Suc n*))))))
by (*simp add*: *lzip.ctr*(*2*))
have *6*: (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* (*n* + *the-enat* (*llength* (*LCons x xs*)))))))) =
(*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + *the-enat* (*llength* (*xs*)))))))
using *eSuc-enat lfinite-LConsI.hyps*(*1*) *llength-eq-infty-conv-lfinite* **by** *force*
have *7*: *lappend* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*LCons x xs*) (*iterates Suc n*))))
(*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* (*n* + *the-enat* (*llength* (*LCons x xs*)))))))) =
lappend (*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*))))))
else (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*))))))
(*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + ((*the-enat* (*llength xs*))))))))

using *5 6* **by** *auto*
show *?thesis* **using** *3 4 7* **by** *auto*
qed
qed

lemma *kfilter-lappend-ltake*:
assumes *k* < *llength xs*
shows *kfilter P n* (*ltake k xs*) = *ltake* (*llength*(*kfilter P n* (*ltake k xs*))) (*kfilter P n xs*)
proof −
have *1*: *kfilter P n xs* = *lappend* (*kfilter P n* (*ltake* (*the-enat k*) *xs*))
(*kfilter P* (*n*+ (*the-enat k*)) (*ldropn* (*the-enat k*) *xs*))
using *kfilter-lappend-lfinite*[*of* (*ltake* (*the-enat k*) *xs*) *P n* (*ldropn* (*the-enat k*) *xs*)]
by (*metis assms enat-iless enat-ord-code*(*4*) *lappend-ltake-ldrop ldrop-enat lfinite-ltake*
llength-ltake min.strict-order-iff neq-iff the-enat.simps)
have *2*: *ltake* (*llength*(*kfilter P n* (*ltake k xs*)))
(*lappend* (*kfilter P n* (*ltake* (*the-enat k*) *xs*))
(*kfilter P* (*n*+ (*the-enat k*)) (*ldropn* (*the-enat k*) *xs*))) =
kfilter P n (*ltake k xs*)
by (*metis assms enat-the-enat lfinite-ltake llength-eq-infty-conv-lfinite llength-ltake*
ltake-all ltake-lappend1 min.strict-order-iff order-refl)
show *?thesis* **using** *1 2* **by** *simp*
qed

lemma *lfilter-ldropn-llength*:
assumes *k* < *llength xs*
shows *llength* (*lfilter P* ((*ldropn k xs*))) ≤ *llength* (*lfilter P* (*xs*))
using *assms*
proof (*induct k arbitrary*: *xs*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*

proof (*cases xs*)
case *LNil*
then show *?thesis* **by** *simp*
next
case (*LCons x21 x22*)
then show *?thesis* **using** *Suc Suc-ile-eq* **by** *auto*
(*meson Suc-ile-eq dual-order.trans ile-eSuc*)
qed
qed

lemma *lfilter-nlength-imp*:
shows *llength* (*lfilter* (λ*x*. *P x* ∧ *Q x*) *xs*) ≤ *llength* (*lfilter P xs*)
proof −
have *0*: *lfilter* (λ*x*. *P x* ∧ *Q x*) *xs* = *lfilter* (λ*x*. *Q x* ∧ *P x*) *xs*
by *meson*
have *1*: *lfilter* (λ*x*. *Q x* ∧ *P x*) *xs* = *lfilter Q* (*lfilter P xs*)
using *lfilter-lfilter*[*of Q P xs*] **by** *auto*
have *2*: *llength* (*lfilter Q* (*lfilter P xs*)) ≤ *llength* (*lfilter P xs*)
using *llength-lfilter-ile* **by** *blast*
show *?thesis* **by** (*simp add*: *1 2 0*)
qed

lemma *lnths-kfilter-lfilter*:
lnths xs (*lset*(*kfilter P 0 xs*)) = *lfilter P xs*
using *lfilter-conv-lnths*[*of P xs*] *kfilter-lset*[*of P 0 xs*]
by *simp*

1.1.9 lrev

lemma *lnull-lrev*[*simp*]:
lnull (*lrev xs*) = (*lnull xs*)
unfolding *lrev-def*
by (*metis* (*full-types*) *llist-of-list-of lnull-llist-of rev.simps*(*1*) *rev-rev-ident*)

lemma *lrev-LNil*[*simp*]:
lrev LNil = *LNil*
unfolding *lrev-def*
by *simp*

lemma *lrev-LCons*[*simp*]:
lrev (*LCons x xs*) = (*if lfinite xs then* (*lappend* (*lrev xs*) (*LCons x LNil*)) *else* (*LCons x xs*))
unfolding *lrev-def*
by *simp*
(*metis lappend-llist-of-llist-of llist-of.simps*(*1*) *llist-of.simps*(*2*))

lemma *ltl-lrev*[*simp*]:
¬*lnull xs* ⟹
ltl (*lrev xs*) =
(*if lfinite xs then*

(*if* (∃ *a*. *xs* = (*LCons a LNil*)) *then LNil else lappend* (*ltl* (*lrev* (*ltl xs*))) (*LCons* (*lhd xs*) *LNil*))
else ltl xs)
using *ltl-lappend*[*of* (*lrev* (*ltl xs*)) (*LCons* (*lhd xs*) *LNil*)]
by (*metis lfinite-ltl lhd-LCons-ltl llist.collapse*(*1*) *llist.disc*(*1*) *lnull-lrev lrev-LCons ltl-simps*(*2*))

lemma *lmap-lrev*:
lmap f (*lrev xs*) = *lrev* (*lmap f xs*)
unfolding *lrev-def*
by (*simp add*: *rev-map*)

lemma *lfinite-lrev* [*simp*]:
lfinite (*lrev xs*) = *lfinite xs*
by (*simp add*: *lrev-def*)

lemma *lset-lrev*:
lset (*lrev xs*) = *lset xs*
unfolding *lrev-def*
by *simp*

lemma *llength-lrev*[*simp*]:
llength (*lrev xs*) = *llength xs*
unfolding *lrev-def*
by (*simp add*: *length-list-of*)

lemma *lrev-llist-of* [*simp*]:
lrev (*llist-of xs*) = *llist-of* (*rev xs*)
unfolding *lrev-def*
by *simp*

lemma *list-of-lrev* [*simp*]:
lfinite xs ⟹ *list-of* (*lrev xs*) = *rev* (*list-of xs*)
unfolding *lrev-def*
by *simp*

lemma *lnth-lrev*:
assumes *lfinite xs*
(*enat i*) < *llength xs*
shows (*lnth* (*lrev xs*) *i*) = (*lnth xs* ((*the-enat* (*llength xs*)) − (*Suc i*)))
using *assms*
unfolding *lrev-def*
by *simp*
(*metis enat-ord-simps*(*2*) *length-list-of length-list-of-conv-the-enat nth-list-of rev-nth*)

lemma *lappend-lrev-lfinite*:
assumes *lfinite xs*
lfinite ys
shows *lrev* (*lappend xs ys*) = *lappend* (*lrev ys*) (*lrev xs*)
using *assms*
unfolding *lrev-def*
by (*simp add*: *lappend-llist-of-llist-of list-of-lappend*)

lemma *lappend-lrev-infa*:
assumes *lfinite xs*
¬*lfinite ys*
shows *lrev* (*lappend xs ys*) = *lappend xs ys*
using *assms*
unfolding *lrev-def* **by** *auto*

lemma *lappend-lrev--infb*:
assumes ¬*lfinite xs*
shows *lrev* (*lappend xs ys*) = *lappend* (*lrev xs*) (*lrev ys*)
using *assms*
unfolding *lrev-def* **by** (*simp add*: *lappend-inf*)

lemma *lrev-lrev*:
lrev (*lrev xs*) = *xs*
by (*simp add*: *lrev-def*)

lemma *lrev-ltake*:
assumes *lfinite xs*
enat k < *llength xs*
shows *lrev* (*ltake k xs*) = *ldrop* (*the-enat*(*llength xs*) −*k*) (*lrev xs*)
using *assms*
unfolding *lrev-def*
by *simp*
(*metis ldrop-llist-of length-list-of-conv-the-enat rev-take take-list-of*)

lemma *lrev-ldrop*:
assumes *lfinite xs*
enat k < *llength xs*
shows *lrev* (*ldrop k xs*) = *ltake* (*the-enat*(*llength xs*) −*k*) (*lrev xs*)
using *assms*
unfolding *lrev-def*
by *simp*
(*metis drop-list-of ldrop-enat length-list-of-conv-the-enat rev-drop*)

lemma *lrev-lsubc*:
assumes *lfinite xs*
enat i ≤ *j*
enat j < *llength xs*
shows *lrev* (*lsubc i j xs*) = *lsubc* ((*the-enat*(*llength xs*)) −(*j*+*1*)) ((*the-enat*(*llength xs*)) − (*i*+*1*)) (*lrev xs*)
proof −
have *1*: *lrev* (*lsubc i j xs*) = *lrev* (*ltake* (*eSuc*(*j*−*i*)) (*ldrop i xs*))
unfolding *lsubc-def min-def* **using** *assms* **apply** *simp*
by (*metis assms*(*2*) *co.enat.exhaust-sel iless-Suc-eq not-less-zero order-le-less-trans*)
have *20*: *i*<∞
by *simp*
have *21*: (*llength* (*ldrop* (*enat i*) *xs*)) = *llength xs* − *i*
by (*simp add*: *ldrop-enat*)

have *22*: (*the-enat* (*llength xs* − *i*)) − (*j* −*i* +*1*) = (*llength xs*) −*eSuc j*
using *assms* **apply** *simp*
using *eSuc-enat llength-eq-infty-conv-lfinite* **by** *fastforce*
have *23*: (*enat* (*the-enat* (*llength* (*ldrop* (*enat i*) *xs*)) − (*j* − *i* + *1*))) =
(*llength xs*) − *eSuc j*
using *21 22* **by** *presburger*
have *2*: *lrev* (*ltake* (*eSuc*(*j*−*i*)) (*ldrop i xs*)) =
ldrop ((*llength xs*) − *eSuc j*) (*lrev* (*ldrop* (*enat i*) *xs*))
using *lrev-ltake*[*of* (*ldrop i xs*) (*j*−*i*)+*1*] **using** *assms 23* **apply** *auto*
by (*metis 21 22 add.commute diff-is-0-eq′ eSuc-enat enat-ord-simps*(*1*) *enat-the-enat ldrop-enat ldropn-0 lfinite-ldrop llength-eq-infty-conv-lfinite ltake-all not-le-imp-less plus-1-eq-Suc*)
have *3*: (*lrev* (*ldrop* (*enat i*) *xs*)) = (*ltake* ((*llength xs*) − *i*) (*lrev xs*))
using *lrev-ldrop*[*of xs i*] **using** *assms order-le-less-trans*
by (*metis enat-the-enat idiff-enat-enat llength-eq-infty-conv-lfinite*)
have *30*: *eSuc j* ≤ *llength xs*
using *assms*(*3*) *ileI1* **by** *blast*
have *31*: *llength* (*lrev* (*ldrop* (*enat i*) *xs*)) = *llength xs* − *i*
by (*simp add*: *21*)
have *32*: *llength* (*ldrop* ((*llength xs*) − *eSuc j*) (*lrev* (*ldrop* (*enat i*) *xs*))) =
(*llength xs* − *i*) − ((*llength xs*) − *eSuc j*)
by (*metis 23 31 ldrop-enat llength-ldropn*)
have *33*: (*llength xs* − *i*) − ((*llength xs*) − *eSuc j*) = (*eSuc j*) − *i*
by (*metis 1 2 32 add.commute assms*(*2*) *assms*(*3*) *eSuc-enat enat-ord-simps*(*1*) *idiff-enat-enat llength-lrev llength-lsubc ordered-cancel-comm-monoid-diff-class.diff-add-assoc2 plus-1-eq-Suc*)

have *4*: *ldrop* ((*llength xs*) − *eSuc j*) (*lrev* (*ldrop* (*enat i*) *xs*)) =
ldrop ((*llength xs*) − *eSuc j*) ((*ltake* ((*llength xs*) − *i*) (*lrev xs*)))
using *3* **by** *presburger*
have *40*: (*min* (*enat* (*the-enat* (*llength xs*) − (*j*+*1*))) (*epred* (*llength* (*lrev xs*)))) =
((*llength xs*) − *eSuc j*)
using *assms*(*1*) *eSuc-enat llength-eq-infty-conv-lfinite* **by** *fastforce*
have *41*: *min* (*enat* (*the-enat* (*llength xs*) − (*j* + *1*))) (*epred* (*llength* (*lrev xs*))) =
(*enat* (*the-enat* (*llength xs*) − (*j* + *1*)))
by (*metis 40 add.commute assms*(*1*) *eSuc-enat enat-the-enat idiff-enat-enat llength-eq-infty-conv-lfinite plus-1-eq-Suc*)
have *42*: *eSuc* (*enat* (*the-enat* (*llength xs*) − (*i* + *1*)) − *enat* (*the-enat* (*llength xs*) − (*j* + *1*))) =
eSuc (*llength xs* − (*i*+*1*) − (*llength xs* − (*j*+*1*)))
using *assms*(*1*) *llength-eq-infty-conv-lfinite* **by** *fastforce*
have *43*: *eSuc* (*llength xs* − (*i*+*1*) − (*llength xs* − (*j*+*1*))) =
eSuc (*j* −*i*)
using *30 assms*(*1*) *eSuc-enat llength-eq-infty-conv-lfinite* **by** *fastforce*
have *44*: *eSuc* (*j* −*i*) + (*enat* (*the-enat* (*llength xs*) − (*j* + *1*))) =
llength xs − *i*
using *30 assms*(*1*) *assms*(*2*) *eSuc-enat llength-eq-infty-conv-lfinite* **by** *fastforce*
have *45*: (*eSuc* (*enat* (*the-enat* (*llength xs*) − (*i* + *1*)) − *enat* (*the-enat* (*llength xs*) − (*j* + *1*))) +
(*enat* (*the-enat* (*llength xs*) − (*j* + *1*)))) =
llength xs − *i*
using *42 43 44* **by** *presburger*
have *5*: *ldrop* ((*llength xs*) − *eSuc j*) ((*ltake* ((*llength xs*) − *i*) (*lrev xs*))) =
lsubc ((*the-enat*(*llength xs*)) − (*j*+*1*)) ((*the-enat*(*llength xs*)) − (*i*+*1*)) (*lrev xs*)

unfolding *lsubc-def ltake-ldrop*
using *40 41 42 43 44* **by** *presburger*
show *?thesis*
using *1 2 4 5* **by** *presburger*
qed

lemma *llast-lrev*:
assumes ¬ *lnull xs*
shows *llast* (*lrev xs*) = (*if lfinite xs then lfirst xs else llast xs*)
using *assms*
unfolding *lrev-def*
by (*simp add*: *last-rev lfirst-def*)

lemma *lfirst-lrev*:
assumes ¬ *lnull xs*
shows *lfirst* (*lrev xs*) = (*if lfinite xs then llast xs else lfirst xs*)
using *assms*
unfolding *lrev-def*
by *simp*
(*metis hd-rev lfirst-def lhd-llist-of llast-llist-of llist-of-list-of*)

lemma *llist-all2-lrevI*:
llist-all2 P xs ys ⟹ *llist-all2 P* (*lrev xs*) (*lrev ys*)
unfolding *lrev-def*
by *simp*
(*metis llist-all2-lfiniteD llist-all2-llist-of llist-of-list-of*)

lemma *llist-all2-lrevD-lfinite*:
assumes *llist-all2 P* (*lrev xs*) (*lrev ys*)
lfinite xs
lfinite ys
shows *llist-all2 P xs ys*
using *assms*
unfolding *lrev-def*
by *simp*
(*metis llist-all2-llist-of llist-of-list-of*)

lemma *llist-all2-lrevD-not-lfinite*:
assumes *llist-all2 P* (*lrev xs*) (*lrev ys*)
¬*lfinite xs*
¬*lfinite ys*
shows *llist-all2 P xs ys*
using *assms* **by** (*simp add*: *lrev-def*)

lemma *llist-all2-lrevD*:
assumes *llist-all2 P* (*lrev xs*) (*lrev ys*)
shows *llist-all2 P xs ys*
by (*metis assms lfinite-lrev llist-all2-lfiniteD llist-all2-lrevD-lfinite llist-all2-lrevD-not-lfinite*)

lemma *llist-all2-lrev*:

llist-all2 P (*lrev xs*) (*lrev ys*) ⟷ *llist-all2 P xs ys*
using *llist-all2-lrevD llist-all2-lrevI* **by** *blast*

1.1.10 Transfer rules

context includes *lifting-syntax*
begin

lemma *lbutlast-transfer* [*transfer-rule*]:
(*llist-all2 A* ===> *llist-all2 A*) *lbutlast lbutlast*
by (*auto simp add*: *rel-fun-def lbutlast-conv-ltake llist-all2-llengthD llist-all2-ltakeI*)

lemma *lleast-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *llist-all2 A* ===> (=)) *lleast lleast*
unfolding *lleast-def*[*abs-def*]
by (*auto simp add*: *rel-fun-def*)
(*metis* (*full-types*, *opaque-lifting*) *llist-all2-conv-all-lnth*)

lemma *lfuse-transfer* [*transfer-rule*]:
(*llist-all2 A* ===> *llist-all2 A* ===> *llist-all2 A*) *lfuse lfuse*
by(*auto simp add*: *rel-fun-def intro*: *llist-all2-lfuseI*)

lemma *ridx-transfer* [*transfer-rule*]:
((*R* ===> *R* ===> (=)) ===> *llist-all2 R* ===> (=)) *ridx ridx*
by (*simp add*: *llist-all2-rsp rel-fun-def ridx-def llist-all2-conv-all-lnth*)
(*meson Suc-ile-eq order-less-imp-le*)

lemma *lsub-transfer* [*transfer-rule*]:
((=) ===> (=) ===> *llist-all2 A* ===> *llist-all2 A*) *lsub lsub*
by (*auto simp add*: *lsub-def rel-fun-def intro*: *llist-all2-ltakeI llist-all2-ldropI*)

lemma *lsubc-transfer* [*transfer-rule*]:
((=) ===> (=) ===> *llist-all2 A* ===> *llist-all2 A*) *lsubc lsubc*
by (*auto simp add*: *lsubc-def rel-fun-def min-def llist-all2-llengthD*
intro: *llist-all2-ltakeI llist-all2-ldropI*)

lemma *lfusecat-transfer* [*transfer-rule*]:
(*llist-all2* (*llist-all2 A*) ===> *llist-all2 A*) *lfusecat lfusecat*
by(*auto intro*: *llist-all2-lfusecatI*)

lemma *lrev-parametric* [*transfer-rule*]:
shows (*llist-all2 A* ===> *llist-all2 A*) *lrev lrev*
by (*rule rel-funI*)(*rule llist-all2-lrevI*)

end

end

1.2 Non-empty coinductive lists

Coinductive lists are formalised by Andreas Lochbihler in [6]. We define coinductive non-empty lists *'a nellist* as a subtype of coinductive lists using the quotient type construction. The usual operators, like take, drop, length, nth, filter etc. are defined for *'a nellist*. The formalisation is based on terminated coinductive list defined by Andreas Lochbihler.

theory *NELList* **imports**
LList-Extras
begin

Coinductive non-empty lists *'a nellist* are the codatatype defined by the constructors *NNil* of type *'a ⇒ 'a nellist* and *NCons* of type *'a ⇒ 'a nellist ⇒ 'a nellist*.

1.2.1 Type definition

consts *nlast0* :: *'a*

codatatype (*nset*: *'a*) *nellist* =
NNil (*nlast* : *'a*)
| *NCons* (*nhd* : *'a*) (*ntl* : *'a nellist*)
for
map: *nmap*
rel: *nellist-all2*
where
nhd (*NNil* -) = *undefined*
| *ntl* (*NNil b*) = *NNil b*
| *nlast* (*NCons* - *nell*) = *nlast0 nell*

overloading
nlast0 == *nlast0*::*'a nellist ⇒ 'a*
begin

partial-function (*tailrec*) *nlast0*
where *nlast0 nell* = (*case nell of* (*NNil x*) ⇒ *x* | (*NCons y nell'*) ⇒ *nlast0 nell'*)

end

lemma *nlast0-nlast* [*simp*]: *nlast0* = *nlast*
proof −
have *1*: ⋀*x*. *nlast0 x* = *nlast x*
by (*simp add*: *nlast0.simps nlast-def*)
show *?thesis* **using** *1* **by** (*rule ext*)
qed

lemmas *nlast-NNil* [*code*, *nitpick-simp*] = *nellist.sel*(*1*)

lemma *nlast-NCons* [*simp*, *code*, *nitpick-simp*]: *nlast* (*NCons x nell*) = *nlast nell*
by *simp*

declare *nellist.sel(2)* [*simp del*]

definition *nfirst* :: *'a nellist ⇒ 'a*
where *nfirst nell* = (*case nell of* (*NNil b*) ⇒ *b* | (*NCons b nell'*) ⇒ *b*)

primcorec *unfold-nellist* :: (*'a ⇒ bool*) ⇒ (*'a ⇒ 'b*) ⇒ (*'a ⇒ 'b*) ⇒ (*'a ⇒ 'a*) ⇒ *'a ⇒ 'b nellist*
where
p a ⟹ *unfold-nellist p g1 g21 g22 a* = *NNil* (*g1 a*) |
- ⟹ *unfold-nellist p g1 g21 g22 a* =
NCons (*g21 a*) (*unfold-nellist p g1 g21 g22* (*g22 a*))

declare
unfold-nellist.ctr(1) [*simp*]
nellist.corec(1) [*simp*]

1.2.2 Code generator setup

More lemmas about generated constants

lemma *ntl-unfold-nellist*:
ntl (*unfold-nellist IS-NNIL NNIL NHD NTL a*) =
(*if IS-NNIL a then NNil* (*NNIL a*) *else unfold-nellist IS-NNIL NNIL NHD NTL* (*NTL a*))
by(*simp*)

lemma *is-NNil-ntl* [*simp*]:
is-NNil nell ⟹ *is-NNil* (*ntl nell*)
by(*cases nell*) *simp-all*

lemma *nlast-ntl* [*simp*]: *nlast* (*ntl nell*) = *nlast nell*
by(*cases nell*) *simp-all*

lemma *unfold-nellist-eq-NNil* [*simp*]:
unfold-nellist IS-NNIL NNIL NHD NTL a = *NNil b* ⟷ *IS-NNIL a* ∧ *b* = *NNIL a*
by(*auto simp add*: *unfold-nellist.code*)

lemma *NNil-eq-unfold-nellist* [*simp*]:
NNil b = *unfold-nellist IS-NNIL NNIL NHD NTL a* ⟷ *IS-NNIL a* ∧ *b* = *NNIL a*
by(*auto simp add*: *unfold-nellist.code*)

lemma *nmap-is-NNil*:
is-NNil nell ⟹ *nmap f nell* = *NNil* (*f* (*nlast nell*))
by(*clarsimp simp add*: *is-NNil-def*)

declare *nellist.map-sel(2)*[*simp*]

lemma *ntl-nmap* [*simp*]:
ntl (*nmap f nell*) = *nmap f* (*ntl nell*)
by(*cases nell*) *simp-all*

lemma *nmap-eq-NNil-conv*:
nmap f nell = *NNil y* ⟷ (∃ *y'*. *nell* = *NNil y'* ∧ *f y'* = *y*)

by(*cases nell*) *simp-all*

lemma *NNil-eq-nmap-conv*:
  $NNil\ y = nmap\ f\ nell \longleftrightarrow (\exists\, y'.\ nell = NNil\ y' \wedge f\ y' = y)$
by(*cases nell*) *auto*

declare *nellist.set-sel*(*1*)[*simp*]

lemma *nset-ntl*: $nset\ (ntl\ nell) \subseteq nset\ nell$
by(*cases nell*) *auto*

lemma *in-nset-ntlD*: $x \in nset\ (ntl\ nell) \Longrightarrow x \in nset\ nell$
using *nset-ntl*[*of nell*] **by** *auto*

theorem *nellist-set-induct*[*consumes 1*, *case-names findnil find step*]:
  assumes $x \in nset\ nell$
  and $\bigwedge nell.\ is\text{-}NNil\ nell \Longrightarrow P\ (nlast\ nell)\ nell$
  and $\bigwedge nell.\ \neg\ is\text{-}NNil\ nell \Longrightarrow P\ (nhd\ nell)\ nell$
  and $\bigwedge nell\ y.\ [\![\neg\ is\text{-}NNil\ nell;\ y \in nset\ (ntl\ nell);\ P\ y\ (ntl\ nell)]\!] \Longrightarrow P\ y\ nell$
  shows $P\ x\ nell$
using *assms*
proof (*induct*)
case (*NNil z*)
then show *?case* **by** *force*
next
case (*NCons1 z1 z2*)
then show *?case* **by** (*fastforce simp del*: *nellist.disc*(*2*) *iff*: *nellist.disc*(*2*))
next
case (*NCons2 z1 z2 xa*)
then show *?case* **by** *auto*
qed

lemma *nset-induct'* [*consumes 1*, *case-names findnil find step*]:
 assumes *major*: $x \in nset\ nell$
 and *0*: $\bigwedge nell.\ is\text{-}NNil\ nell \Longrightarrow P\ (nlast\ nell)$
 and *1*: $\bigwedge nell.\ P\ (NCons\ x\ nell)$
 and *2*: $\bigwedge\ x'\ nell.\ [\![\ x \in nset\ nell;\ P\ nell\ ]\!] \Longrightarrow P\ (NCons\ x'\ nell)$
shows $P\ nell$
using *major*
proof (*induct y*≡*x nell rule*: *nellist-set-induct*)
case (*findnil nell*)
then show *?case* **using** *0 1 2* **by** (*metis nellist.collapse*(*1*) *nellist.map-disc-iff nellist.map-sel*(*1*))
next
case (*find nell*)
then show *?case* **by** (*metis 1 nellist.collapse*(*2*))
next
case (*step nell*)
then show *?case* **by** (*metis 2 nellist.collapse*(*2*))
qed

lemma *nset-induct* [*consumes 1*, *case-names findnil find step*, *induct set*: *nset*]:
assumes *major*: $x \in$ *nset nell*
and *findnil*: $\bigwedge$*nell*. *is-NNil nell* $\Longrightarrow$ *P* (*nlast nell*)
and *find*: $\bigwedge$*nell*. *P* (*NCons x nell*)
and *step*: $\bigwedge$ *x' nell*. $[\![x \in$ *nset nell*; $x \neq x'$; *P nell* $]\!] \Longrightarrow$ *P* (*NCons x' nell*)
shows *P nell*
using *major*
proof (*induct rule*: *nset-induct'*)
case (*findnil nell*)
then show *?case* **by** (*simp add*: *assms*(*2*))
next
case (*find nell*)
then show *?case* **by** (*simp add*: *assms*(*3*))
next
case (*step x' nell*)
then show *?case* **by** (*metis assms*(*4*) *find*)
qed

1.2.3 Connection with *'a llist*

primcorec *llist-of-nellist* :: *'a nellist* ⇒ *'a llist*
where *llist-of-nellist nell* = (*case nell of NNil b* ⇒ *LCons b LNil* |
NCons x nell' ⇒ *LCons x* (*llist-of-nellist nell'*))

context
fixes
b :: *'a*
begin
primcorec *nellist-of-llist-a* :: *'a llist* ⇒ *'a nellist* **where**
nellist-of-llist-a ll = (*case ll of LNil* ⇒ *NNil b* |
LCons x ll' ⇒ *NCons x* (*nellist-of-llist-a ll'*))
end

abbreviation *nellist-of-llist* == (λ *ll*. *nellist-of-llist-a* (*llast ll*) (*lbutlast ll*))

simps-of-case *nellist-of-llist-a-simps* [*simp*, *code*, *nitpick-simp*]: *nellist-of-llist-a.code*

lemmas *nellist-of-llist-a-LNil* = *nellist-of-llist-a-simps*(*1*)
and *nellist-of-llist-a-LCons* = *nellist-of-llist-a-simps*(*2*)

lemma *nlast-nellist-of-llist-a-lnull* [*simp*]:
lnull ll $\Longrightarrow$ *nlast* (*nellist-of-llist-a b ll*) = *b*
unfolding *lnull-def* **by** *simp*

declare *nellist-of-llist-a.sel*(*1*)[*simp del*]

lemma *lhd-LNil*:
lhd LNil = *undefined*
by(*simp add*: *lhd-def*)

lemma *nhd-NNil*:
nhd (*NNil b*) = *undefined*
by(*simp add*: *nhd-def*)

lemma *nhd-nellist-of-llist-a* [*simp*]:
nhd (*nellist-of-llist-a b ll*) = *lhd ll*
by (*cases ll*)
(*simp-all add*: *lhd-LNil nhd-NNil*)

lemma *ntl-nellist-of-llist-a* [*simp*]:
ntl (*nellist-of-llist-a b ll*) = *nellist-of-llist-a b* (*ltl ll*)
by(*cases ll*) *simp-all*

lemma *llist-of-nellist-eq-LNil*:
llist-of-nellist nell = *LCons* (*nlast nell*) *LNil* ⟷ *is-NNil nell*
by (*simp add*: *nellist.case-eq-if llist-of-nellist.code*)

simps-of-case *llist-of-nellist-simps* [*simp*, *code*, *nitpick-simp*]: *llist-of-nellist.code*

lemmas *llist-of-nellist-NNil* = *llist-of-nellist-simps*(*1*)
and *llist-of-nellist-NCons* = *llist-of-nellist-simps*(*2*)

declare *llist-of-nellist.sel* [*simp del*]

lemma *lhd-llist-of-nellist* [*simp*]:
¬ *is-NNil nell* ⟹ *lhd* (*llist-of-nellist nell*) = *nhd nell*
by(*cases nell*) *simp-all*

lemma *lhd-llist-of-nellist1* [*simp*]:
is-NNil nell ⟹ *lhd* (*llist-of-nellist nell*) = *nlast nell*
by (*cases nell*) *simp-all*

lemma *lhd-llist-of-nellist2* [*simp*]:
(*case nell of* (*NNil b*) ⇒ *lhd LNil* | (*NCons b nell'*) ⇒ *lhd* (*llist-of-nellist nell*)) = *nhd nell*
by (*cases nell*) (*simp-all add*: *lhd-LNil nhd-NNil*)

lemma *ltl-llist-of-nellist* [*simp*]:
¬ *is-NNil nell* ⟹ *ltl* (*llist-of-nellist nell*) = *llist-of-nellist* (*ntl nell*)
by(*cases nell*) *simp-all*

lemma *ltl-llist-of-nellist1* [*simp*]:
is-NNil nell ⟹ *ltl* (*llist-of-nellist nell*) = *LNil*
by(*cases nell*) *simp-all*

lemma *ltl-llist-of-nellist2* [*simp*]:
(*case nell of* (*NNil b*) ⇒ (*LCons b LNil*) |
(*NCons b nell'*) ⇒ *ltl* (*llist-of-nellist nell*)) = *llist-of-nellist* (*ntl nell*)
by (*simp add*: *llist-of-nellist.code nellist.case-eq-if*)

lemma *nellist-of-llist-a-cong* [*cong*]:
  assumes *ll* = *ll′ lfinite ll′* ⟹ *b* = *b′*
  shows *nellist-of-llist-a b ll* = *nellist-of-llist-a b′ ll′*
proof(*unfold* ‹*ll* = *ll′*›)
  from *assms* **have** *lfinite ll′* ⟶ *b* = *b′* **by** *simp*
  thus *nellist-of-llist-a b ll′* = *nellist-of-llist-a b′ ll′*
  by(*coinduction arbitrary*: *ll′*) *auto*
qed

primcorec *snocn* :: *′a nellist* ⇒ *′a* ⇒ *′a nellist*
 where *snocn nell a* =
        (*case nell of* (*NNil x*) ⇒ *NCons x* (*NNil a*) |
              (*NCons x nell′*) ⇒ *NCons x* (*snocn nell′ a*))

simps-of-case *snocn-code* [*code*, *simp*, *nitpick-simp*]: *snocn.code*

lemma *snocn-simps* [*simp*]:
 shows *nhd-snocn*: *nhd*(*snocn nell a*) = *nfirst nell*
 and *ntl-snocn*: *ntl*(*snocn nell a*) = (*if is-NNil nell then* (*NNil a*) *else snocn* (*ntl nell*) *a*)
by (*case-tac* [!] *nell*)
   (*auto simp add*: *nfirst-def*)

lemma *is-NNil-snocn*:
 is-NNil(*snocn nell a*) ⟷ *False*
by (*auto simp add*: *snocn-def*)

lemma *nmap-snocn-distrib*:
  nmap f (*snocn nell a*) = *snocn* (*nmap f nell*) (*f a*)
proof (*coinduction arbitrary*: *nell rule*: *nellist.coinduct-strong*)
case (*Eq-nellist nella*)
then show *?case*
by (*auto simp add*: *nellist.case-eq-if nellist.map-sel*(*1*))
qed

definition *nfinite* :: *′a nellist* ⇒ *bool*
where *nfinite nell* ≡ *lfinite* (*llist-of-nellist nell*)

lemma *nfinite-induct* [*consumes 1*, *case-names NNil NCons*]:
  assumes *nfinite nell*
  and ⋀*y*. *P* (*NNil y*)
  and ⋀*x nell*. ⟦*nfinite nell*; *P nell*⟧ ⟹ *P* (*NCons x nell*)
  shows *P nell*
using *assms*
unfolding *nfinite-def*
proof (*induct ll*≡*llist-of-nellist nell arbitrary*: *nell rule*: *lfinite-induct*)
case *LNil*
then show *?case* **by** *simp*
next
case *LCons*

then show *?case* **by** (*metis lfinite.cases lnull-def ltl-llist-of-nellist ltl-simps*(*2*)
nellist.collapse(*1*) *nellist.exhaust-sel*)
qed

lemma
shows *nfinite-NNil*: *nfinite* (*NNil x*)
and *nfinite-NConsI*: *nfinite nell* ⟹ *nfinite* (*NCons x nell*)
unfolding *nfinite-def*
by *auto*

declare *nfinite-NNil* [*iff*]

lemma *is-NNil-imp-nfinite* [*simp*]:
is-NNil nell ⟹ *nfinite nell*
using *lfinite.simps llist-of-nellist-eq-LNil* **by** (*auto simp add*: *nfinite-def*)

lemma *nfinite-NCons* [*simp*]:
nfinite (*NCons x nell*) = *nfinite nell*
by (*simp add*: *nfinite-def*)

lemma *nfinite-ntl* [*simp*]:
nfinite (*ntl nell*) = *nfinite nell*
by (*cases nell*) *simp-all*

lemma *nfinite-code* [*code*]:
nfinite (*NNil x*) = *True*
nfinite (*NCons x nell*) = *nfinite nell*
by *simp-all*

lemma *nfinite-imp-finite-nset*:
assumes *nfinite nell*
shows *finite* (*nset nell*)
using *assms*
by (*induct nell rule*:*nfinite-induct*) *simp-all*

lemma *nfinite-snocn* [*simp*]:
nfinite(*snocn nell a*) ⟷ *nfinite nell*
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs* **thus** *?rhs*
proof (*induct zs*≡*snocn nell a arbitrary*: *nell rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis is-NNil-snocn nellist.disc*(*1*))
next
case (*NCons x nell*)
then show *?case*
by (*cases nell*) *simp-all*

qed
next
assume *?rhs* **thus** *?lhs*
by (*induct rule*: *nfinite-induct*) *auto*
qed

lemma *snocn-inf*:
¬ *nfinite nell* ⟹ *snocn nell a* = *nell*
proof (*coinduction arbitrary*: *nell*)
case (*Eq-nellist nella*)
then show *?case*
proof −
have *1*: *is-NNil* (*snocn nella a*) = *is-NNil nella*
using *Eq-nellist* **by** *auto*
have *2*: (*is-NNil* (*snocn nella a*) ⟶ *is-NNil nella* ⟶ *nlast* (*snocn nella a*) = *nlast nella*)
by *auto*
have *3*: (¬ *is-NNil* (*snocn nella a*) ⟶ ¬ *is-NNil nella* ⟶
nhd (*snocn nella a*) = *nhd nella* ∧
(∃ *nell*. *ntl* (*snocn nella a*) = *snocn nell a* ∧ *ntl nella* = *nell* ∧ ¬ *nfinite nell*))
by (*simp add*: *Eq-nellist nellist.case-eq-if*)
from *1 2 3* **show** *?thesis* **by** *blast*
qed
qed

lemma *nfinite-nmap* [*simp*]:
nfinite (*nmap f nell*) = *nfinite nell* (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs* **thus** *?rhs*
proof (*induct zs*≡*nmap f nell arbitrary*: *nell rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*metis nellist.disc*(*1*) *nellist.map-disc-iff is-NNil-imp-nfinite*)
next
case (*NCons x nell*)
then show *?case* **by** (*metis nellist.sel*(*5*) *nfinite-ntl ntl-nmap*)
qed
next
assume *?rhs* **thus** *?lhs*
by (*induct rule*: *nfinite-induct*) *simp-all*
qed

lemma *nset-snocn-nfinite* [*simp*]:
nfinite nell ⟹ *nset*(*snocn nell a*) = *nset nell* ∪ {*a*}
by (*induct rule*: *nfinite-induct*) *auto*

lemma *nset-snocn1*:
nset (*snocn nell a*) ⊆ *nset nell* ∪ {*a*}
proof (*cases nfinite nell*)
case *True*
then show *?thesis* **by** *simp*
next

case *False*
then show *?thesis* **by** (*auto simp add*: *snocn-inf*)
qed

lemma *nset-snocn-conv*:
nset (*snocn nell a*) = (*if nfinite nell then nset nell* ∪ {*a*} *else nset nell*)
by (*simp add*: *snocn-inf*)

lemma *in-nset-snocn-iff*:
x ∈ *nset* (*snocn nell a*) ⟷ *x* ∈ *nset nell* ∨ *nfinite nell* ∧ *x* = *a*
by (*metis Un-iff empty-iff insert-iff nset-snocn-conv*)

lemma *llist-of-nellist-inverse-1*:
assumes ¬ *nfinite nell*
shows *nellist-of-llist-a b* (*llist-of-nellist nell*) = *snocn nell b*
using *assms*
proof (*coinduction arbitrary*: *nell*)
case (*Eq-nellist nella*)
then show *?case*
proof −
have *1*: *is-NNil* (*nellist-of-llist-a b* (*llist-of-nellist nella*)) = *is-NNil* (*snocn nella b*)
by *auto*
have *2*: (*is-NNil* (*nellist-of-llist-a b* (*llist-of-nellist nella*)) ⟶
is-NNil (*snocn nella b*) ⟶
nlast (*nellist-of-llist-a b* (*llist-of-nellist nella*)) = *nlast* (*snocn nella b*))
by *simp*
have *3*: (¬ *is-NNil* (*nellist-of-llist-a b* (*llist-of-nellist nella*)) ⟶
¬ *is-NNil* (*snocn nella b*) ⟶
nhd (*nellist-of-llist-a b* (*llist-of-nellist nella*)) = *nhd* (*snocn nella b*) ∧
(∃ *nell*.
ntl (*nellist-of-llist-a b* (*llist-of-nellist nella*)) =
nellist-of-llist-a b (*llist-of-nellist nell*) ∧
ntl (*snocn nella b*) = *snocn nell b* ∧ ¬ *nfinite nell*))
by (*metis Eq-nellist lhd-llist-of-nellist ltl-llist-of-nellist nfinite-ntl*
nhd-nellist-of-llist-a ntl-nellist-of-llist-a snocn-inf)
from *1 2 3* **show** *?thesis* **by** *blast*
qed
qed

lemma *llist-of-nellist-inverse-2*:
assumes *nfinite nell*
shows *nellist-of-llist-a b* (*llist-of-nellist nell*) = *snocn nell b*
using *assms*
by (*induct rule*: *nfinite-induct*) *simp-all*

lemma *llist-of-nellist-inverse* [*simp*]:
shows *nellist-of-llist-a b* (*llist-of-nellist nell*) = *snocn nell b*
using *llist-of-nellist-inverse-1 llist-of-nellist-inverse-2* **by** *fastforce*

lemma *llist-of-nellist-inverse-3*:

assumes ¬ *nfinite nell*
shows *nellist-of-llist-a* (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) = *nell*
using *assms*
proof (*coinduction arbitrary*: *nell*)
case (*Eq-nellist nella*)
then show *?case*
proof −
have *1*: *is-NNil* (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) =
is-NNil nella
by (*metis Eq-nellist is-NNil-imp-nfinite lbutlast.disc*(*2*) *llist-of-nellist.disc-iff*
ltl-llist-of-nellist nellist-of-llist-a.disc(*2*))
have *2*: (*is-NNil* (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) ⟶
is-NNil nella ⟶
nlast (*nellist-of-llist-a* (*nlast nella*)
(*lbutlast* (*llist-of-nellist nella*))) = *nlast nella*)
using *Eq-nellist is-NNil-imp-nfinite* **by** *blast*
have *3*: (¬ *is-NNil* (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) ⟶
¬ *is-NNil nella* ⟶
nhd (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) =
nhd nella ∧
(∃ *nell*.
ntl (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) =
nellist-of-llist-a (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) ∧
ntl nella = *nell* ∧ ¬ *nfinite nell*))
by (*auto simp add*: *llist.case-eq-if Eq-nellist*)
from *1 2 3* **show** *?thesis* **by** *blast*
qed
qed

lemma *llist-of-nellist-inverse-4*:
assumes *nfinite nell*
shows *nellist-of-llist-a* (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) = *nell*
using *assms*
by (*induct rule*: *nfinite-induct*) (*simp-all add*: *llist-of-nellist.code*)

lemma *llist-of-nellist-inverse-a* [*simp*]:
shows *nellist-of-llist-a* (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) = *nell*
using *llist-of-nellist-inverse-3 llist-of-nellist-inverse-4* **by** *fastforce*

lemma *nlast-llast*:
assumes *nfinite nell*
shows *nlast nell* = *llast*(*llist-of-nellist nell*)
using *assms*
by (*induct rule*: *nfinite-induct*)
(*simp-all add*: *llist-of-nellist.code*)

lemma *llist-of-nellist-inverse-b* [*simp*]:
shows *nellist-of-llist* (*llist-of-nellist nell*) = *nell*
by (*metis lappend-inf lbutlast-snoc llist-of-nellist-inverse llist-of-nellist-inverse-a*
nfinite-def snocn-inf nlast-llast)

lemma *nellist-of-llist-a-eq* [*simp*]:
nellist-of-llist-a b′ ll = *NNil b* ⟷ *b* = *b′* ∧ *ll* = *LNil*
by(*cases ll*) *auto*

lemma *NNil-eq-nellist-of-llist-a* [*simp*]:
NNil b = *nellist-of-llist-a b′ ll* ⟷ *b* = *b′* ∧ *ll* = *LNil*
by(*cases ll*) *auto*

lemma *nellist-of-llist-a-inject* [*simp*]:
nellist-of-llist-a b llx = *nellist-of-llist-a c lly* ⟷ *llx* = *lly* ∧ (*lfinite lly* ⟶ *b* = *c*)
(**is** *?lhs* ⟷ *?rhs*)
proof(*intro iffI conjI impI*)
assume *?rhs*
thus *?lhs* **by**(*auto intro*: *nellist-of-llist-a-cong*)
next
assume *?lhs*
thus *llx* = *lly*
by(*coinduction arbitrary*: *llx lly*)(*auto simp add*: *lnull-def neq-LNil-conv*)
assume *lfinite lly*
thus *b* = *c* **using** ‹*?lhs*›
unfolding ‹*llx* = *lly*› **by**(*induct*) *simp-all*
qed

lemma *nellist-of-llist-a-inverse-1*:
assumes ¬ *lfinite ll*
shows *llist-of-nellist* (*nellist-of-llist-a b ll*) = *lappend ll* (*LCons b LNil*)
using *assms*
by (*coinduction arbitrary*: *ll*) *auto*

lemma *nellist-of-llist-a-inverse-2*:
assumes *lfinite ll*
shows *llist-of-nellist* (*nellist-of-llist-a b ll*) = *lappend ll* (*LCons b LNil*)
using *assms*
proof (*induct ll rule*: *lfinite-induct*)
case (*LNil xs*)
then show *?case* **by** (*simp add*: *lnull-def*)
next
case (*LCons xs*)
then show *?case*
by (*metis nellist-of-llist-a.disc*(*2*) *lappend-code*(*2*) *lhd-LCons-ltl lhd-llist-of-nellist*
llist-of-nellist.code llist-of-nellist.simps(*2*) *llist-of-nellist.simps*(*3*)
ltl-llist-of-nellist nhd-nellist-of-llist-a ntl-nellist-of-llist-a)
qed

lemma *nellist-of-llist-a-inverse* [*simp*]:
shows *llist-of-nellist* (*nellist-of-llist-a b ll*) = *lappend ll* (*LCons b LNil*)
using *nellist-of-llist-a-inverse-1 nellist-of-llist-a-inverse-2* **by** *metis*

lemma *nellist-of-llist-inverse* [*simp*]:

assumes ¬ *lnull ll*
shows *llist-of-nellist* (*nellist-of-llist ll*) = *ll*
using *assms* **by** *simp*

lemma *nmap-nellist-of-llist-a*:
nmap f (*nellist-of-llist-a b ll*) = *nellist-of-llist-a* (*f b*) (*lmap f ll*)
by (*coinduction arbitrary*: *ll*) (*auto simp add*: *nmap-is-NNil*)

lemma *nmap-nellist-of-llist*:
assumes ¬ *lnull ll*
shows *nmap f* (*nellist-of-llist ll*) = *nellist-of-llist* (*lmap f ll*)
using *assms*
by (*metis lappend-inf lbutlast-snoc lfinite-lmap llast-lmap lmap-lbutlast nellist-of-llist-a-cong*
nmap-nellist-of-llist-a)

lemma *lmap-llist-of-nellist*:
lmap f (*llist-of-nellist nell*) = *llist-of-nellist* (*nmap f nell*)
by (*metis llist.map-disc-iff llist-of-nellist.disc-iff llist-of-nellist-inverse-b*
nellist-of-llist-inverse nmap-nellist-of-llist)

definition *cr-nellist* :: *'a llist* ⇒ *'a nellist* ⇒ *bool*
where *cr-nellist* = (λ *ll nell*. *llist-of-nellist nell* = *ll*)

lemma *llist-of-nellist-not-lnull*:
¬ (*lnull* (*llist-of-nellist nell*))
by *simp*

lemma *not-lnull-eq-lappend-lbutlast-llast*:
¬(*lnull ll*) ⟷ *ll* = *lappend* (*lbutlast ll*) (*LCons* (*llast ll*) *LNil*)
using *llist.collapse*(*1*) **by** *fastforce*

lemma *Domainp-help*:
¬ *lnull ll* ⟹ ∃ *nell*. *llist-of-nellist nell* = *ll*
using *nellist-of-llist-inverse* **by** *blast*

lemma *not-lnull-conv-llist-of-nellist*:
¬ *lnull ll* ⟷ (∃ *nell*. *llist-of-nellist nell* = *ll*)
using *Domainp-help llist-of-nellist-not-lnull* **by** *blast*

lemma *Domainp-cr-nellist* [*transfer-domain-rule*]:
Domainp cr-nellist = (λ*ll*. ¬(*lnull ll*))
unfolding *cr-nellist-def Domainp-iff*[*abs-def*]
using *Domainp-help* **by** *fastforce*

lemma *bi-unique-cr-nellist-help*:
llist-of-nellist nelly = *llist-of-nellist nellz* ⟹ *nelly* = *nellz*
by (*coinduction arbitrary*: *nelly nellz*)
(*metis llist-of-nellist-inverse-b*)

lemma *quotient-help-2*:
(¬ *lnull* (*llist-of-nellist nell*) ∧ *nellist-of-llist* (*llist-of-nellist nell*) = *nell*)
by (*simp add*: *bi-unique-cr-nellist-help*)

lemma *quotient-help-nellist-1*:
cr-nellist ll nell ⟶ *nellist-of-llist ll* = *nell*
by (*metis cr-nellist-def llist-of-nellist-inverse-b*)

lemma *quotient-help-nellist-2*:
(*cr-nellist* (*llist-of-nellist nell*) *nell*)
by (*simp add*: *cr-nellist-def*)

lemma *quotient-help-3-nellist*:
((¬ *lnull llx* ∧ *llx* = *lly*) =
(*cr-nellist llx* (*nellist-of-llist llx*) ∧
cr-nellist lly (*nellist-of-llist lly*) ∧
nellist-of-llist llx =
nellist-of-llist lly))
by (*metis Domainp.DomainI Domainp-cr-nellist cr-nellist-def nellist-of-llist-inverse*)

lemma *Quotient-nellist*:
Quotient (λ *llx lly*. ¬ *lnull llx* ∧ *llx* = *lly*)
nellist-of-llist llist-of-nellist cr-nellist
unfolding *Quotient-alt-def*
using *quotient-help-3-nellist quotient-help-nellist-1 quotient-help-nellist-2* **by** *blast*

setup-lifting *Quotient-nellist*

context includes *lifting-syntax*
begin

lemma *bi-unique-cr-nellist* [*transfer-rule*]:
bi-unique cr-nellist
unfolding *cr-nellist-def bi-unique-def*
by (*auto simp add*: *bi-unique-cr-nellist-help*)

lemma *right-total-cr-nellist* [*transfer-rule*]:
right-total cr-nellist
unfolding *cr-nellist-def right-total-def*
by *simp*

lemma *NNil-transfer* [*transfer-rule*]:
(*A* ===> *pcr-nellist A*) (λ*b*. *LCons b LNil*) *NNil*
by (*auto simp add*: *pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *NCons-transfer* [*transfer-rule*]:
(*A* ===> *pcr-nellist A* ===> *pcr-nellist A*) *LCons NCons*
by (*auto simp add*: *pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *nmap-transfer* [*transfer-rule*]:
((=) ===> *pcr-nellist* (=) ===> *pcr-nellist* (=)) *lmap nmap*
by (*auto simp add*: *cr-nellist-def nellist.pcr-cr-eq rel-fun-def lmap-llist-of-nellist*)

lemma *is-NNil-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *is-lfirst is-NNil*
by (*simp add*: *cr-nellist-def nellist.pcr-cr-eq rel-fun-def*)
(*metis lbutlast-simps*(*2*) *llast-singleton llist.disc*(*2*) *llist-of-nellist-eq-LNil*
llist-of-nellist-inverse-a nellist-of-llist-inverse)

lemma *is-NNil-nellist-of-llist-conv-is-lfirst*:
assumes ¬ *lnull lx*
shows *is-NNil*(*nellist-of-llist lx*) ⟷ *is-lfirst lx*
using *assms*
by (*cases lx*)
(*simp*, *metis lbutlast.ctr*(*1*) *lbutlast.disc-iff*(*2*) *lbutlast-eq-LNil-conv llist.disc*(*1*)
nellist-of-llist-a.disc-iff(*2*))

lemma *nfirst-transfer-a* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *lhd nfirst*
by (*simp add*: *cr-nellist-def nellist.pcr-cr-eq nfirst-def rel-fun-def llist-of-nellist.simps*(*2*))

lemma *nhd-transfer-a1* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) (λ *ll*. *if is-lfirst ll then lhd LNil else lhd ll*) *nhd*
by (*simp add*: *cr-nellist-def nellist.pcr-cr-eq rel-fun-def OO-def*)
(*metis lbutlast-simps*(*2*) *lhd-llist-of-nellist llist-of-nellist-eq-LNil nhd-nellist-of-llist-a*
quotient-help-2)

lemma *ntl-transfer* [*transfer-rule*]:
(*pcr-nellist A* ===> *pcr-nellist A*) (λ *ll*. *if is-lfirst ll then ll else ltl ll*) *ntl*
proof (*auto simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def intro*!: *llist-all2-ltlI*
lfinite-LConsI dest: *llist-all2-lnullD*)
show ⋀*b y*. *llist-all2 A* (*LCons b LNil*) (*llist-of-nellist y*) ⟹
llist-all2 A (*LCons b LNil*) (*llist-of-nellist* (*ntl y*))
using *llist-all2-ltlI*
by (*metis eq-LConsD is-NNil-ntl llist-all2-LNil1 llist-of-nellist.code llist-of-nellist.simps*(*3*)
llist-of-nellist-eq-LNil ltl-llist-of-nellist nlast-ntl)
next
show ⋀*x y*. ∀ *b*. *x* ≠ *LCons b LNil* ⟹ *llist-all2 A x* (*llist-of-nellist y*) ⟹
llist-all2 A (*ltl x*) (*llist-of-nellist* (*ntl y*))
by (*metis lhd-LCons-ltl llist-all2-LNil2 llist-all2-lnullD llist-all2-ltlI*
llist-of-nellist.disc-iff ltl-llist-of-nellist ltl-llist-of-nellist1)
qed

lemma *nfinite-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *lfinite nfinite*
by (*auto simp add*: *nellist.pcr-cr-eq cr-nellist-def nfinite-def rel-fun-def*)

lemma *llist-of-nellist-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *id llist-of-nellist*
by (*simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)

lemma *nellist-of-llist-a-transfer* [*transfer-rule*]:
((=) ===> (=) ===> *pcr-nellist* (=)) (λ*b ll*. *lappend ll* (*LCons b LNil*)) *nellist-of-llist-a*
by (*auto simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def*)

lemma *nlast-nellist-of-llist-a-lfinite* [*simp*]:
lfinite ll ⟹ *nlast* (*nellist-of-llist-a b ll*) = *b*
by(*induct rule*: *lfinite.induct*) *simp-all*

lemma *snocn-transfer* [*transfer-rule*]:
(*pcr-nellist* (*A*) ===> (*A*) ===> *pcr-nellist* (*A*)) (λ *ll a*. *lappend ll* (*LCons a LNil*)) *snocn*
unfolding *rel-fun-def*
by (*auto simp add*: *pcr-nellist-def nellist.pcr-cr-eq cr-nellist-def OO-def*)
(*metis LNil-transfer llist.rel-intros*(*2*) *llist-all2-lappendI llist-of-nellist-inverse*
nellist-of-llist-a-inverse)

lemma *nellist-all2-help-a*:
llist-all2 P (*llist-of-nellist nella*) (*llist-of-nellist nellb*) ⟹ *nellist-all2 P nella nellb*
by (*coinduction arbitrary*: *nella nellb*)
(*metis lhd-llist-of-nellist llist.disc*(*1*) *llist-all2-LCons-LCons llist-all2-LNil1*
llist-all2-LNil2 llist-all2-lhdD llist-all2-ltlI llist-of-nellist.disc-iff
llist-of-nellist-eq-LNil ltl-llist-of-nellist ltl-simps(*2*))

lemma *nellist-all2-help-b*:
nellist-all2 P nella nellb ⟹ *llist-all2 P* (*llist-of-nellist nella*) (*llist-of-nellist nellb*)
proof (*coinduction arbitrary*: *nella nellb*)
case *LNil*
then show *?case*
by *simp*
next
case *LCons*
then show *?case*
by *auto*
(*metis llist-of-nellist.simps*(*2*) *nellist.case-eq-if nellist.rel-sel*,
metis llist-all2-LNil1 ltl-llist-of-nellist ltl-llist-of-nellist1 nellist.rel-sel)
qed

lemma *nellist-all2-transfer* [*transfer-rule*]:
((=) ===> *pcr-nellist* (=) ===> *pcr-nellist* (=) ===> (=)) *llist-all2 nellist-all2*
unfolding *nellist.pcr-cr-eq cr-nellist-def*
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*

lift-definition *nappend* :: *'a nellist* ⇒ *'a nellist* ⇒ *'a nellist*
is (λ *llx lly*. *lappend llx lly*)
by *auto*

lemma *llist-all2-llist-of-nellist-1*:
assumes ¬ *lnull y*
llist-all2 (λ*x z*. *llist-of-nellist z* = *x*) *llist1 y*
llist-all2 (λ*x z*. *llist-of-nellist z* = *x*) *llist2 y*
shows *llist1* = *llist2*
proof −
have *1*: *llist-all2* (λ*x z*. *llist-of-nellist z* = *x*) *llist1 y* =
llist-all2 (=) *llist1* (*lmap llist-of-nellist y*)
using *llist-all2-lmap2*[*of* (=) *llist1 llist-of-nellist y*]
using *llist-all2-mono* **by** *fastforce*
have *2*: *llist-all2* (λ*x z*. *llist-of-nellist z* = *x*) *llist2 y* =
llist-all2 (=) *llist2* (*lmap llist-of-nellist y*)
using *llist-all2-lmap2*[*of* (=) *llist2 llist-of-nellist y*]
using *llist-all2-mono* **by** *fastforce*
show *?thesis* **using** *assms*
by (*metis* (*full-types*) *1 2 llist.rel-eq*)
qed

lemma *llist-all2-llist-of-nellist-2*:
assumes ¬ *lnull y*
llist-all2 (λ*z x*. *llist-of-nellist z* = *x*) *y llist1*
llist-all2 (λ*z x*. *llist-of-nellist z* = *x*) *y llist2*
shows *llist1* = *llist2*
proof −
have *1*: *llist-all2* (λ*z x*. *llist-of-nellist z* = *x*) *y llist1* =
llist-all2 (=) (*lmap llist-of-nellist y*) *llist1*
using *llist-all2-lmap1*[*of* (=)]
using *llist-all2-mono* **by** *fastforce*
have *2*: *llist-all2* (λ*z x*. *llist-of-nellist z* = *x*) *y llist2* =
llist-all2 (=) (*lmap llist-of-nellist y*) *llist2*
using *llist-all2-lmap1*[*of* (=)]
using *llist-all2-mono* **by** *fastforce*
show *?thesis* **using** *assms*
by (*metis* (*full-types*) *1 2 llist.rel-eq*)
qed

lift-definition *nconcat* :: *'a nellist nellist* ⇒ *'a nellist*
is (λ *lxs*. *lconcat lxs*)
apply (*simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)
using *llist.rel-cases mem-Collect-eq llist-all2-llist-of-nellist-1* **by** *fastforce*

lift-definition *nfilter* :: (*'a* ⇒ *bool*) ⇒ *'a nellist* ⇒ *'a nellist*
is λ *P ll*. (*if lnull*(*lfilter P ll*) *then ll else lfilter P ll*)
by *auto*

lift-definition *lappendn* :: *'a llist* ⇒ *'a nellist* ⇒ *'a nellist*
is *lappend*
by *auto*

lift-definition *nzip* :: *'a nellist* ⇒ *'b nellist* ⇒ (*'a* × *'b*) *nellist*
is (λ *llx lly. lzip llx lly*)
by *auto*

lift-definition *niterates* :: (*'a* ⇒ *'a*) ⇒ *'a* ⇒ *'a nellist*
is (λ *f a . iterates f a*)
by *auto*

lift-definition *ndistinct* :: *'a nellist* ⇒ *bool*
is *ldistinct*
by *auto*

lift-definition *nnth* :: *'a nellist* ⇒ *nat* ⇒ *'a*
is λ *ll n. lnth ll* (*the-enat* (*min* (*enat n*) ((*epred* (*llength ll*)))))
by *blast*

lift-definition *nlength* :: *'a nellist* ⇒ *enat*
is λ *ll. epred*(*llength ll*)
by *auto*

lift-definition *ndropn* :: *nat* ⇒ *'a nellist* ⇒ *'a nellist*
is λ *n ll.* *ldropn* (*the-enat* (*min* (*enat n*) ((*epred*(*llength ll*))))) *ll*
by *auto*
(*metis co.enat.exhaust-sel enat-the-enat iless-Suc-eq infinity-ileE leD llength-eq-0*
min.cobounded1 min.cobounded2)

lift-definition *ntake* :: *enat* ⇒ *'a nellist* ⇒ *'a nellist*
is λ *n ll. ltake* (*eSuc n*) *ll*
by *auto*

lift-definition *ntaken* :: *nat* ⇒ *'a nellist* ⇒ *'a nellist*
is λ *n ll. ltake* (*Suc n*) *ll*
using *enat-0-iff*(*2*) **by** *auto*

1.2.4 The nlast element *nlast*

lemma *nlast-not-nfinite*:
assumes ¬ *nfinite nell*
shows *nlast nell* = *undefined*
unfolding *nlast0-nlast*[*symmetric*]
using *assms*
by (*rule contrapos-np*)
(*induct nell rule*: *nlast0.raw-induct*[*rotated 1*, *OF refl*, *consumes 1*],
auto split: *nellist.split-asm*)

lemma *nlast-nellist-of-llist-a*:
nlast (*nellist-of-llist-a y ll*) = (*if lfinite ll then y else undefined*)
by (*simp add*: *nfinite-def nlast-not-nfinite*)

lemma *nlast-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) (λ*ll*. *if lfinite ll then llast ll else undefined*) *nlast*
by (*auto simp add*: *cr-nellist-def pcr-nellist-def nlast-nellist-of-llist-a OO-def*
dest: *llist-all2-lfiniteD*)
(*simp add*: *llist.rel-eq nfinite-def nlast-llast nlast-not-nfinite rel-funI*)

lemma *nlast-nmap* [*simp*]:
nfinite nell ⟹ *nlast* (*nmap f nell*) = *f* (*nlast nell*)
by (*induct rule*: *nfinite-induct*)
(*auto simp add*: *nellist.map-sel*(*1*))

lemma *nset-nlast*:
nfinite nell ⟹ *nlast nell* ∈ *nset nell*
by (*induct rule*: *nfinite-induct*)
(*simp-all add*: *nellist.set-sel*(*3*))

1.2.5 *nset*

lemma *lset-llist-of-nellist-1*:
assumes *nfinite nell*
shows *lset* (*lbutlast* (*llist-of-nellist nell*)) ∪ {*nlast nell*}= *nset nell* (**is** *?lhs* = *?rhs*)
proof(*intro set-eqI iffI*)
fix *x*
assume *x* ∈ *?lhs*
thus *x* ∈ *?rhs*
proof −
have *1*: *nlast nell* = *x* ⟹ *x* ∈ *nset nell*
using *assms nset-nlast* **by** *blast*
have *2*: *nlast nell* = *x* ⟹ *x* ∈ *nset nell*
unfolding *nlast0-nlast*[*symmetric*] **by** (*simp add*: *1*)
have *3*: *x* ∈ *lset* (*lbutlast*(*llist-of-nellist nell*)) ⟹ *x* ∈ *nset nell*
proof (*induct lbutlast* (*llist-of-nellist nell*) *arbitrary*: *nell rule*: *llist-set-induct*)
case *find*
then show *?case*
by (*metis lbutlast.disc*(*1*) *llist.disc*(*1*) *llist-of-nellist-inverse-b ltl-llist-of-nellist1*
nellist.set-sel(*2*) *nhd-nellist-of-llist-a*)
next
case (*step y*)
then show *?case*
by (*metis lbutlast.disc*(*1*) *lbutlast-ltl llist.disc*(*1*) *ltl-llist-of-nellist ltl-llist-of-nellist1*
nellist.disc(*1*) *nellist.exhaust-sel nellist.set-intros*(*3*))
qed
show *?thesis*
using *2 3* ‹*x* ∈ *lset* (*lbutlast* (*llist-of-nellist nell*)) ∪ {*nlast nell*}› **by** *blast*
qed
next
fix *x*
assume *x* ∈ *?rhs*
thus *x* ∈ *?lhs*
proof(*induct rule*: *nellist-set-induct*)

```
    case (findnil nell)
    then show ?case
     by (cases nell) auto
    next
    case (find nell)
    thus ?case
     by (metis UnI1 lbutlast.disc-iff(2) llist.set-sel(1) ltl-llist-of-nellist
         nhd-nellist-of-llist-a quotient-help-2)
    next
    case step
    thus ?case
     by (metis Un-iff in-lset-ltlD lbutlast-ltl ltl-llist-of-nellist nlast-ntl)
  qed
qed

lemma lset-llist-of-nellist-2:
assumes ¬nfinite nell
shows  lset (lbutlast (llist-of-nellist nell)) = nset nell (is ?lhs = ?rhs)
proof(intro set-eqI iffI)
  fix x
  assume x ∈ ?lhs
  thus x ∈ ?rhs
    proof −
     have 3: x ∈ lset (lbutlast (llist-of-nellist nell)) ⟹ x ∈ nset nell
       proof (induct lbutlast (llist-of-nellist nell) arbitrary: nell rule: llist-set-induct)
       case find
       then show ?case
        by (metis lbutlast.disc(1) llist.disc(1) llist-of-nellist-inverse-a ltl-llist-of-nellist1
            nellist.set-sel(2) nhd-nellist-of-llist-a)
       next
       case (step y)
       then show ?case
         by (metis lbutlast.disc(1) lbutlast-ltl llist.disc(1) ltl-llist-of-nellist
             ltl-llist-of-nellist1 nellist.collapse(2) nellist.set-intros(3))
       qed
     show ?thesis
       using  3 ‹x ∈ lset (lbutlast (llist-of-nellist nell))› by blast
  qed
next
  fix x
  assume x ∈ ?rhs
  thus x ∈ ?lhs
  using assms
  proof(induct rule: nellist-set-induct)
    case (findnil nell)
    then show ?case
     by (cases nell) auto
    next
    case (find nell)
    thus ?case
```

by (*metis lbutlast-not-lfinite lhd-llist-of-nellist llist.set-sel*(*1*) *llist-of-nellist-not-lnull nfinite-def*)
next
case *step*
thus *?case*
by (*metis in-lset-ltlD lbutlast-ltl ltl-llist-of-nellist nfinite-ntl*)
qed
qed

lemma *lset-llist-of-nellist* [*simp*]:
(*if nfinite nell then lset* (*lbutlast*(*llist-of-nellist nell*)) ∪ {*nlast nell*}
else lset (*lbutlast* (*llist-of-nellist nell*))) = *nset nell*
using *lset-llist-of-nellist-1 lset-llist-of-nellist-2* **by** *auto*

lemma *lset-llist-of-nellist-a* [*simp*]:
lset(*llist-of-nellist nell*) = *nset nell*
proof (*cases nfinite nell*)
case *True*
then show *?thesis*
by (*metis lappend-lbutlast-llast-id-lfinite lbutlast-lfinite llist.simps*(*19*)
llist-of-nellist-not-lnull lset-LNil lset-lappend-lfinite lset-llist-of-nellist-1 nfinite-def nlast-llast)
next
case *False*
then show *?thesis* **by** (*metis lbutlast-not-lfinite lset-llist-of-nellist-2 nfinite-def*)
qed

lemma *nset-nellist-of-llist-a* [*simp*]:
shows *nset* (*nellist-of-llist-a b ll*) = (*if lfinite ll then lset ll* ∪ {*b*} *else lset ll*)
proof (*cases lfinite ll*)
case *True*
then show *?thesis*
by (*metis llist.simps*(*19*) *lset-LNil lset-lappend-lfinite lset-llist-of-nellist-a nellist-of-llist-a-inverse*)
next
case *False*
then show *?thesis*
by (*metis lappend-inf lset-llist-of-nellist-a nellist-of-llist-a-inverse*)
qed

lemma *nset-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *lset nset*
by(*auto simp add*: *cr-nellist-def nellist.pcr-cr-eq*)

end

1.2.6 *nmap*

lemma *nmap-eq-NCons-conv*:
nmap f nellx = *NCons y nelly* ⟷

$(\exists\, z\ nellz.\ nellx = NCons\ z\ nellz \wedge f\ z = y \wedge nmap\ f\ nellz = nelly)$
by(*cases nellx*) *simp-all*

lemma *NCons-eq-nmap-conv*:
$NCons\ y\ nelly = nmap\ f\ \ nellx \longleftrightarrow$
$(\exists\, z\ nellz.\ nellx = NCons\ z\ nellz \wedge f\ z = y \wedge nmap\ f\ nellz = nelly)$
by(*cases nellx*) *auto*

1.2.7 Appending two nonempty lazy lists *nappend*

lemma *nappend-NNil* [*simp*, *code*, *nitpick-simp*]:
$nappend\ (NNil\ b)\ nell = (NCons\ b\ nell)$
by *transfer auto*

lemma *nappend-NCons* [*simp*, *code*, *nitpick-simp*]:
$nappend\ (NCons\ a\ nellx)\ nelly = NCons\ a\ (nappend\ nellx\ nelly)$
by *transfer auto*

lemma *nhd-nappend* [*simp*]:
$nhd(nappend\ nellx\ nelly) = (if\ is\text{-}NNil\ nellx\ then\ nlast\ nellx\ else\ nhd\ nellx)$
by (*cases nellx*) *auto*

lemma *ntl-nappend* [*simp*]:
$ntl(nappend\ nellx\ nelly) = (if\ is\text{-}NNil\ nellx\ then\ nelly\ else\ nappend\ (ntl\ nellx)\ nelly)$
by (*cases nellx*) *auto*

lemma *is-NNil-nappend*:
$is\text{-}NNil(nappend\ nellx\ nelly) \longleftrightarrow False$
by (*cases nellx*) *auto*

lemma *nappend-assoc*:
$nappend\ (nappend\ nellx\ nelly)\ nellz = nappend\ nellx\ (nappend\ nelly\ nellz)$
by *transfer* (*auto simp add*: *split-beta lappend-assoc*)

lemma *nmap-nappend-distrib*:
$nmap\ f\ (nappend\ nellx\ nelly) = nappend\ (nmap\ f\ nellx)\ (nmap\ f\ nelly)$
by *transfer* (*auto simp add*: *split-beta lmap-lappend-distrib*)

lemma *nlast-nappend*:
$nlast\ (nappend\ nellx\ nelly) = (if\ nfinite\ nellx\ then\ nlast\ nelly\ else\ nlast\ nellx)$
by *transfer* (*auto simp add*: *llast-lappend*)

lemma *nfinite-nappend*:
$nfinite\ (nappend\ nellx\ nelly) \longleftrightarrow nfinite\ nellx \wedge nfinite\ nelly$
by *transfer auto*

lemma *nappend-inf*:
$\neg\ nfinite\ nellx \Longrightarrow nappend\ nellx\ nelly = nellx$
by *transfer* (*auto simp add*: *lappend-inf*)

lemma *nappend-snocn-inf*:
assumes $\neg$ *nfinite nell*
shows *nappend nell* (*NNil a*) = *snocn nell a*
using *assms*
by (*simp add*: *nappend-inf snocn-inf*)

lemma *nappend-snocn-finite*:
assumes *nfinite nell*
shows *nappend nell* (*NNil a*) = *snocn nell a*
using *assms*
by (*induct rule*: *nfinite-induct*) *simp-all*

lemma *nappend-snocn*:
nappend nell (*NNil a*) = *snocn nell a*
by (*meson nappend-snocn-finite nappend-snocn-inf*)

lemma *split-nellist-first*:
assumes *x* ∈ *nset nell*
shows *nell* = (*NNil x*) ∨ (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ∨
(∃ *ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys* ∧ *x* ∉ *nset ys*) ∨
(∃ *ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys* ∧ *x* ∉ *nset ys*)
using *assms*
by *transfer*
(*auto*,
metis eq-LConsD lhd-lappend llist.disc(*1*) *llist.expand split-llist-first*)

lemma *split-nellist*:
assumes *x* ∈ *nset nell*
shows *nell* = (*NNil x*) ∨ (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ∨
(∃ *ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys*) ∨
(∃ *ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys*)
using *assms*
by (*meson split-nellist-first*)

lemma *split-nellist-a*:
assumes *nell* = (*NNil x*) ∨ (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ∨
(∃ *ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys*) ∨
(∃ *ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys*)
shows *x* ∈ *nset nell*
proof −
have *1*: *nell* = (*NNil x*) ⟹ *x* ∈ *nset nell*
by *simp*
have *2*: (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ⟹ *x* ∈ *nset nell*
by *auto*
have *3*: (∃ *ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys*) ⟹ *x* ∈ *nset nell*
by *transfer auto*
have *4*: (∃ *ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys*) ⟹ *x* ∈ *nset nell*
by *transfer auto*
show *?thesis* **using** *1 2 3 4 assms* **by** *blast*
qed

1.2.8 Appending a nonempty lazy list to a lazy list *lappendn*

lemma *lappendn-LNil* [*simp*, *code*, *nitpick-simp*]:
lappendn LNil nell = nell
by *transfer auto*

lemma *lappendn-LCons* [*simp*, *code*, *nitpick-simp*]:
lappendn (LCons x ll) nell = NCons x (lappendn ll nell)
by *transfer auto*

lemma *nlast-lappendn-lfinite* [*simp*]:
lfinite ll ⟹ nlast (lappendn ll nell) = nlast nell
by *transfer*
(auto simp add: llast-lappend)

lemma *nset-lappendn-lfinite* [*simp*]:
lfinite ll ⟹ nset (lappendn ll nell) = lset ll ∪ nset nell
by *transfer auto*

lemma *nlength-nappend* [*simp*]:
nlength (nappend nellx nelly) = nlength nellx + nlength nelly +1
by *transfer*
(auto, metis co.enat.exhaust-sel epred-iadd1 iadd-Suc-right llength-eq-0 plus-1-eSuc(2))

lemma *nfinite-nlength-enat*:
assumes *nfinite nell*
shows *∃ n. nlength nell = enat n*
using *assms*
by *transfer (metis epred-conv-minus idiff-enat-enat lfinite-llength-enat one-enat-def)*

lemma *nlength-eq-enat-nfiniteD*:
nlength nell = enat n ⟹ nfinite nell
by *transfer (metis epred-Infty llength-eq-enat-lfiniteD not-lfinite-llength)*

lemma *nfinite-conv-nlength-enat*:
nfinite nell ⟷ (∃ n. nlength nell = enat n)
using *nfinite-nlength-enat nlength-eq-enat-nfiniteD* **by** *blast*

1.2.9 The length of a nonempty lazy list *nlength*

lemma [*simp*, *nitpick-simp*]:
shows *nlength-NNil*: *nlength (NNil b) = 0*
and *nlength-NCons*: *nlength (NCons x nell) = eSuc (nlength nell)*
by (*transfer*, *simp*) (*transfer*, *auto*)

lemma *llength-llist-of-nellist* [*simp*]:
epred(llength (llist-of-nellist nell)) = nlength nell
by *transfer auto*

lemma *nlength-nmap* [*simp*]:
nlength (nmap f nell) = nlength nell

by *transfer simp*

definition *gen-nlength* :: *nat* ⇒ *'a nellist* ⇒ *enat*
where *gen-nlength n nell* = *enat n* + *nlength nell*

lemma *gen-nlength-code* [*code*]:
gen-nlength n (*NNil b*) = *enat n*
gen-nlength n (*NCons x nell*) = *gen-nlength* (*n* + *1*) *nell*
by (*simp-all add*: *gen-nlength-def iadd-Suc eSuc-enat*[*symmetric*] *iadd-Suc-right*)

lemma *nlength-code* [*code*]:
nlength = *gen-nlength 0*
by(*simp add*: *gen-nlength-def fun-eq-iff zero-enat-def*)

1.2.10 The nth element of a nonempty lazy list *nnth*

lemma *nnth-NNil* [*nitpick-simp*]:
nnth (*NNil b*) *n* = *b*
by *transfer simp*

lemma *nnth-NCons*:
nnth (*NCons x nell*) *n* = (*case n of 0* ⇒ *x* | *Suc n'* ⇒ *nnth nell n'*)
by (*transfer fixing*: *n*)
(*auto simp add*: *lnth-LCons Nitpick.case-nat-unfold zero-enat-def min-enat1-conv-enat*,
metis enat-0-iff(*1*) *less-not-refl3 llength-eq-0 min-def min-enat1-conv-enat*,
metis enat-min-eq-0-iff min-enat1-conv-enat not-gr-zero the-enat.simps the-enat-0,
metis One-nat-def epred-enat epred-min min-enat1-conv-enat the-enat.simps)

lemma *nnth-code* [*simp*, *nitpick-simp*, *code*]:
shows *nnth-0*: *nnth* (*NCons x nell*) *0* = *x*
and *nnth-Suc-NCons*: *nnth* (*NCons x nell*) (*Suc n*) = *nnth nell n*
by(*simp-all add*: *nnth-NCons*)

lemma *lnth-llist-of-nellist* [*simp*]:
lnth (*llist-of-nellist nell*) (*the-enat* (*min* (*enat n*) ((*epred*(*llength* (*llist-of-nellist nell*))))))
= *nnth nell n*
by *transfer auto*

lemma *nnth-nmap* [*simp*]:
enat n ≤ *nlength nell* ⟹ *nnth* (*nmap f nell*) *n* = *f* (*nnth nell n*)
by *transfer*
(*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lmap lnth-lmap min.orderE*
the-enat.simps)

lemma *nhd-conv-nnth*:
¬ *is-NNil nell* ⟹ *nhd nell* = *nnth nell 0*
by (*metis nellist.collapse*(*2*) *nnth-0*)

lemmas *nnth-0-conv-nhd* = *nhd-conv-nnth*[*symmetric*]

lemma *nnth-ntl*:
nnth (*ntl nell*) *n* = *nnth nell* (*Suc n*)
by (*metis nellist.exhaust-sel nellist.sel*(*4*) *nnth-NNil nnth-Suc-NCons*)

lemma *in-nset-conv-nnth*:
x ∈ *nset nell* ⟷ (∃ *n*. *enat n* ≤ *nlength nell* ∧ *nnth nell n* = *x*)
by *transfer*
(*metis eSuc-epred iless-Suc-eq in-lset-conv-lnth llength-eq-0 min-absorb1 the-enat.simps*)

lemma *nnth-beyond*:
nlength nell < *enat n* ⟹ *nnth nell n* = *nlast nell*
by *transfer*
(*metis co.enat.exhaust-sel epred-llength less-enatE lfinite-ltl llast-conv-lnth llength-eq-0*
llength-eq-enat-lfiniteD min.absorb4 the-enat.simps)

lemma *exists-Pred-nnth-nset*:
(∃ *x* ∈ *nset nell*. *P x*) = (∃ *n*. *n*≤ *nlength nell* ∧ *P* (*nnth nell n*))
by (*metis in-nset-conv-nnth*)

lemma *nset-conv-nnth*:
nset nell = {*nnth nell n*| *n*. *enat n* ≤ *nlength nell*}
by (*auto simp add*: *in-nset-conv-nnth*)

lemma *nnth-nappend1*:
enat n ≤ *nlength nellx* ⟹ *nnth* (*nappend nellx nelly*) *n* = *nnth nellx n*
proof (*induct n arbitrary*: *nellx*)
case *0*
then show *?case*
by (*metis is-NNil-def is-NNil-nappend nellist.sel*(*1*) *nhd-nappend nnth-0-conv-nhd nnth-NNil*)
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis*
using *Suc.prems enat-0-iff*(*1*) **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis*
using *Suc.hyps Suc.prems Suc-ile-eq* **by** *auto*
qed
qed

lemma *nnth-nappend2*:
⟦*nlength nellx* = *enat k*; *k*<*n*⟧ ⟹ *nnth* (*nappend nellx nelly*) *n* = *nnth nelly* (*n* −*Suc k*)
proof (*induct n arbitrary*: *nellx k*)
case *0*
then show *?case* **by** *blast*
next
case (*Suc n*)

then show *?case*
by (*cases nellx*)
(*auto simp add*: *eSuc-def zero-enat-def split*: *enat.split-asm*)
qed

lemma *nnth-nappend*:
nnth (*nappend nellx nelly*) *n* =
(*if enat n* ≤ *nlength nellx then nnth nellx n else nnth nelly* (*n*− *Suc*(*the-enat*(*nlength nellx*))))
by (*cases nlength nellx*)
(*auto simp add*: *nnth-nappend1 nnth-nappend2*)

lemma *nnth-nlast*:
nfinite nell ⟹ *nlast nell* = *nnth nell* (*the-enat* (*nlength nell*))
by *transfer*
(*simp*,
metis co.enat.exhaust-sel enat-the-enat ile-eSuc infinity-ileE lfinite-llength-enat
llast-conv-lnth llength-eq-0 min.idem)

1.2.11 *ntake*

lemma *ntake-NNil* [*simp*, *code*, *nitpick-simp*]:
ntake n (*NNil b*) = (*NNil b*)
by *transfer auto*

lemma *ntake-0* [*simp*]:
ntake 0 nell = (*NNil* (*nfirst nell*))
by *transfer* (*auto simp add*: *ltake.ctr*(*2*))

lemma *ntake-Suc-NCons* [*simp*]:
ntake (*eSuc n*) (*NCons x nell*) = (*NCons x* (*ntake n nell*))
by *transfer auto*

lemma *ntake-Suc*:
ntake (*eSuc n*) *nell* =
(*case nell of* (*NNil b*) ⇒ (*NNil b*) | (*NCons x nell*′) ⇒ (*NCons x* (*ntake n nell*′)))
by (*cases nell*) *simp-all*

lemma *is-NNil-ntake* [*simp*]:
is-NNil(*ntake n nell*) ⟷ *is-NNil nell* ∨ *n=0*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x nell1*)
then show *?thesis*
proof (*cases n*)
case (*enat nat*)
then show *?thesis*
by (*metis NCons enat-coexhaust nellist.disc*(*1*) *nellist.disc*(*2*) *ntake-0 ntake-Suc-NCons*)
next

case *infinity*
then show *?thesis*
by (*metis NCons eSuc-infinity i0-ne-infinity nellist.disc(2) ntake-Suc-NCons*)
qed
qed
lemma *ntake-eq-NNil-iff* [*simp*]:
ntake n nell = (*NNil x*) ⟷ *nell* = (*NNil x*) ∨ (*n* = *0* ∧ *nfirst nell* = *x*)
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
using *ntake-0* **by** *fastforce*
next
case (*NCons x nell1*)
then show *?thesis*
proof (*cases n*)
case (*enat nat*)
then show *?thesis*
by (*metis NCons is-NNil-ntake nellist.disc(1) nellist.disc(2) nellist.inject(1) ntake-0*)
next
case *infinity*
then show *?thesis*
by (*metis NCons eSuc-infinity infinity-ne-i0 nellist.distinct(1) ntake-Suc-NCons*)
qed
qed

lemma *NNil-eq-ntake-iff* [*simp*]:
(*NNil x*) = *ntake n nell* ⟷ *nell* = (*NNil x*) ∨ (*n* = *0* ∧ *nfirst nell* = *x*)
by (*metis ntake-eq-NNil-iff*)

lemma *ntake-NCons* [*code*, *nitpick-simp*]:
ntake n (*NCons x nell*) = (*case n of 0* ⇒ (*NNil x*) | (*eSuc n′*) ⇒ (*NCons x* (*ntake n′ nell*)))
by (*simp add*: *co.enat.case-eq-if*)
(*metis eSuc-epred nellist.simps(6) nfirst-def ntake-Suc-NCons*)

lemma *nhd-ntake* [*simp*]:
n ≠ *0* ⟹ *nhd*(*ntake n nell*) = *nhd nell*
unfolding *nhd-def*
by (*simp add*: *nellist.case-eq-if*)
(*metis* (*no-types*, *lifting*) *co.enat.case-eq-if nellist.collapse(2) nellist.sel(3) ntake-NCons*)

lemma *ntl-ntake*:
n ≠ *0* ⟹ *ntl*(*ntake n nell*) = *ntake* (*epred n*) (*ntl nell*)
by (*cases nell*) (*simp*, *metis eSuc-epred nellist.sel(5) ntake-Suc-NCons*)

lemma *ntl-ntake-0*:
ntl(*ntake 0 nell*) = (*NNil* (*nfirst nell*))
by *simp*

lemma *ntake-ntl*:
ntake n (*ntl nell*) = *ntl*(*ntake* (*Suc n*) *nell*)

by (*simp add*: *enat-0-iff*(*1*) *ntl-ntake*)

lemma *nlength-ntake* [*simp*]:
nlength (*ntake n nell*) = *min n* (*nlength nell*)
by *transfer simp*

lemma *ntake-nmap* [*simp*]:
ntake n (*nmap f nell*) = *nmap f* (*ntake n nell*)
by *transfer simp*

lemma *ntake-ntake* [*simp*]:
ntake n (*ntake m nell*) = *ntake* (*min n m*) *nell*
by *transfer simp*

lemma *nset-ntake*:
nset (*ntake n nell*) ⊆ *nset nell*
by *transfer* (*simp add*: *lset-ltake*)

lemma *ntake-all*:
nlength nell ≤ *m* ⟹ *ntake m nell* = *nell*
by *transfer* (*auto*, *metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all*)

lemma *nfinite-ntake* [*simp*]:
nfinite (*ntake n nell*) ⟷ *nfinite nell* ∨ *n* < ∞
by *transfer* (*metis Extended-Nat.eSuc-mono eSuc-infinity lfinite-ltake*)

lemma *ntake-nappend1*:
n ≤ *nlength nellx* ⟹ *ntake n* (*nappend nellx nelly*) = *ntake n nellx*
by *transfer* (*auto*, *metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1*)

lemma *ntake-nappend2*:
assumes *nlength nellx* < *n*
shows *ntake n* (*nappend nellx nelly*) = *nappend nellx* (*ntake* (*n* − *nlength nellx* −*1*) *nelly*)
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis* **using** *assms*
by (*metis eSuc-le-iff eSuc-minus-1 idiff-0-right ileI1 nappend-NNil nlength-NNil ntake-Suc-NCons*)
next
case (*NCons x21 x22*)
then show *?thesis*
proof (*cases nfinite nellx*)
case *True*
then show *?thesis*
using *assms*
proof (*transfer*)
fix *llxa* :: *'a llist*
fix *na*
fix *llya* :: *'a llist*
assume *a0*: ¬ *lnull llxa* ∧ *llxa* = *llxa*
assume *a1*: *lfinite llxa*

```
    assume a2: epred (llength llxa) < na
    assume a3: ¬ lnull llya ∧ llya = llya
    show ¬ lnull (ltake (eSuc na) (lappend llxa llya)) ∧
      ltake (eSuc na) (lappend llxa llya) =
      lappend llxa (ltake (eSuc (na − epred (llength llxa) − 1)) llya)
    proof −
     have 1: ¬ lnull (ltake (eSuc na) (lappend llxa llya))
       using a0 by force
     have 2:  ltake (eSuc na) (lappend llxa llya) =
              lappend (ltake (eSuc na) llxa) (ltake ((eSuc na) − llength llxa) llya)
       by (meson ltake-lappend)
     have 21: llength llxa ≤ (eSuc na)
       by (metis a0 a2 co.enat.exhaust-sel eSuc-ile-mono leD le-cases llength-eq-0)
     have 3: lappend (ltake (eSuc na) llxa) (ltake ((eSuc na) − llength llxa) llya) =
             lappend llxa (ltake (eSuc na − llength llxa) llya)
       using ltake-lappend2[of llxa (eSuc na) llya] 21 2 by auto
     have 4: (eSuc na − llength llxa) = (eSuc (na − epred (llength llxa) − 1))
       by (metis a0 a1 a2 canonically-ordered-monoid-add-class.lessE co.enat.exhaust-sel eSuc-infinity
           eSuc-minus-1 eSuc-minus-eSuc enat-add-sub-same llength-eq-0 llength-eq-infty-conv-lfinite)
     have 5: (ltake (eSuc na − llength llxa) llya) =
             (ltake (eSuc (na − epred (llength llxa) − 1)) llya)
       using 4 by auto
     show ?thesis using 1 2 3 5 by fastforce
    qed
   qed
   next
   case False
   then show ?thesis using assms
   by (simp add: nappend-inf ntake-all)
  qed
qed

lemma ntake-eq-ntake-antimono:
  ⟦ ntake n nellx = ntake n nelly; m ≤ n ⟧ ⟹ ntake m nellx = ntake m nelly
by (metis min.orderE ntake-ntake)

lemma ntake-nnth:
 assumes  enat m ≤ n
 shows    (nnth (ntake n nell) m) = (nnth nell m)
using assms
proof (induct m arbitrary: nell n)
 case 0
 then show ?case
  proof (cases n rule: enat-coexhaust)
   case 0
   then show ?thesis
   using 0.prems
   by (metis nellist.case(2) nellist.collapse(1) nellist.exhaust-sel nfirst-def
       nnth-0-conv-nhd nnth-NNil ntake-eq-NNil-iff)
   next
```

case (*eSuc n'*)
then show *?thesis*
by (*simp add*: *nellist.case-eq-if nnth-0-conv-nhd ntake-Suc*)
qed
next
case (*Suc m*)
then show *?case*
proof (*cases n rule*: *enat-coexhaust*)
case *0*
then show *?thesis*
using *Suc.prems* **by** (*simp add*: *enat-0-iff*(*1*))
next
case (*eSuc n'*)
then show *?thesis*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
using *Suc.hyps Suc.prems Suc-ile-eq eSuc* **by** *force*
qed
qed
qed

1.2.12 *ntaken*

lemma *ntaken-NNil* [*simp*, *code*, *nitpick-simp*]:
ntaken n (*NNil b*) = (*NNil b*)
by *transfer*
(*metis eSuc-enat llist.disc*(*2*) *ltake-LNil ltake-eSuc-LCons*)

lemma *ntaken-0* [*simp*]:
ntaken 0 nell = (*NNil* (*nfirst nell*))
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** (*metis ntake-0 ntake-NNil ntaken-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis*
by *transfer* (*simp*, (*metis One-nat-def ltake-0 ltake-eSuc-LCons one-eSuc one-enat-def zero-neq-one*))
qed

lemma *ntaken-Suc-NCons* [*simp*]:
ntaken (*Suc n*) (*NCons x nell*) = (*NCons x* (*ntaken n nell*))
by *transfer* (*auto simp add*: *zero-enat-def*, *metis eSuc-enat ltake-eSuc-LCons*)

lemma *ntaken-Suc*:
ntaken (*Suc n*) *nell* =
(*case nell of* (*NNil b*) ⇒ (*NNil b*) | (*NCons x nell'*) ⇒ (*NCons x* (*ntaken n nell'*)))

by (*cases nell*) *simp-all*

lemma *is-NNil-ntaken* [*simp*]:
  is-NNil(*ntaken n nell*) ⟷ *is-NNil nell* ∨ *n=0*
proof (*cases nell*)
 case (*NNil x1*)
 then show *?thesis*
 by *simp*
 next
 case (*NCons x nell1*)
 then show *?thesis*
 proof (*cases n*)
  case *0*
  then show *?thesis*
  by *simp*
  next
  case (*Suc nat*)
  then show *?thesis*
  by (*simp add*: *NCons*)
 qed
qed

lemma *ntaken-eq-NNil-iff* [*simp*]:
  ntaken n nell = (*NNil x*) ⟷ *nell* = (*NNil x*) ∨ (*n* = *0* ∧ *nfirst nell* = *x*)
proof (*cases nell*)
 case (*NNil x1*)
 then show *?thesis*
 by (*metis ntake-0 ntake-NNil ntaken-NNil*)
 next
 case (*NCons x nell1*)
 then show *?thesis*
 proof (*cases n*)
 case *0*
 then show *?thesis*
 by (*simp add*: *NCons*)
 next
 case (*Suc nat*)
 then show *?thesis*
 using *NCons* **by** *force*
 qed
qed

lemma *NNil-eq-ntaken-iff* [*simp*]:
  (*NNil x*) = *ntaken n nell* ⟷ *nell* = (*NNil x*) ∨ (*n* = *0* ∧ *nfirst nell* = *x*)
by (*metis ntaken-eq-NNil-iff*)

lemma *ntaken-NCons* [*code*, *nitpick-simp*]:
  ntaken n (*NCons x nell*) = (*case n of 0* ⇒ (*NNil x*) | (*Suc n′*) ⇒ (*NCons x* (*ntaken n′ nell*)))
by (*cases n*) (*auto simp add*: *nfirst-def*)

lemma *nhd-ntaken* [*simp*]:
n ≠ *0* ⟹ *nhd*(*ntaken n nell*) = *nhd nell*
by (*cases nell*)
(*simp-all add*: *Nitpick.case-nat-unfold ntaken-NCons*)

lemma *ntl-ntaken*:
n ≠ *0* ⟹ *ntl*(*ntaken n nell*) = *ntaken* (*n*−*1*) (*ntl nell*)
by *simp-all*
(*metis Suc-pred nellist.exhaust-sel nellist.sel*(*4*) *nellist.sel*(*5*) *ntaken-NNil ntaken-Suc-NCons*)

lemma *ntl-ntaken-0*:
ntl(*ntaken 0 nell*) = (*NNil* (*nfirst nell*))
by *simp*

lemma *ntaken-ntl*:
ntaken n (*ntl nell*) = *ntl*(*ntaken* (*Suc n*) *nell*)
by (*simp add*: *enat-0-iff*(*1*) *ntl-ntaken*)

lemma *ntaken-nlength* [*simp*]:
nlength (*ntaken n nell*) = *min n* (*nlength nell*)
by *transfer simp*

lemma *ntaken-nmap* [*simp*]:
ntaken n (*nmap f nell*) = *nmap f* (*ntaken n nell*)
using *enat-0-iff*(*2*) **by** *transfer auto*

lemma *ntaken-ntaken* [*simp*]:
ntaken n (*ntaken m nell*) = *ntaken* (*min n m*) *nell*
using *enat-0-iff*(*2*) **by** *transfer auto*

lemma *nset-ntaken*:
nset (*ntaken n nell*) ⊆ *nset nell*
by *transfer* (*simp add*: *lset-ltake*)

lemma *ntaken-all*:
nlength nell ≤ *m* ⟹ *ntaken m nell* = *nell*
by *transfer*
(*auto simp add*: *zero-enat-def*, *metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all*)

lemma *ntaken-nnth*:
shows (*nnth* (*ntaken m nell*) *k*) = (*nnth nell* (*min k m*))
apply *transfer*
by (*auto simp add*: *min-def lnth-ltake ltake-all*)
(*metis co.enat.sel*(*2*) *enat-eSuc-iff enat-ord-simps*(*1*) *epred-le-epredI order-trans*,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0,
meson dual-order.strict-iff-order enat-ord-simps(*2*) *linorder-not-less order-less-le-trans*,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0)

lemma *nfinite-ntaken* [*simp*]:
nfinite (*ntaken n nell*)
by *transfer simp*

lemma *ntaken-nlast*:
nlast (*ntaken n nell*) = *nnth nell n*
using *nnth-nlast*[*of ntaken n nell*] *ntaken-nlength*[*of n nell*]
by (*metis min.absorb3 min.idem min.orderI nfinite-ntaken nnth-beyond not-less-iff-gr-or-eq*
ntaken-all ntaken-nnth the-enat.simps)

lemma *ntaken-nfirst*:
nfirst (*ntaken n nell*) = *nfirst nell*
by *transfer* (*simp add*: *enat-0-iff*(*1*))

lemma *ntaken-nappend1*:
n ≤ *nlength nellx* ⟹ *ntaken n* (*nappend nellx nelly*) = *ntaken n nellx*
by *transfer*
(*simp add*: *zero-enat-def*, *metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1*)

lemma *ntaken-nappend2*:
nlength nellx < (*enat n*) ⟹
ntaken n (*nappend nellx nelly*) = *nappend nellx* (*ntaken* (*n* − (*the-enat*(*nlength nellx*)) −*1*) *nelly*)
proof (*induct n arbitrary*: *nellx nelly*)
case *0*
then show *?case* **using** *zero-enat-def* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis*
by *simp*
next
case (*NCons x21 x22*)
then show *?thesis* **using** *Suc* **by** *simp*
(*metis Extended-Nat.eSuc-mono eSuc-enat enat-ord-code*(*4*) *order-less-imp-not-less the-enat-eSuc*)
qed
qed

lemma *ntaken-eq-ntaken-antimono*:
⟦ *ntaken n nellx* = *ntaken n nelly*; *m* ≤ *n* ⟧ ⟹ *ntaken m nellx* = *ntaken m nelly*
by (*metis min.orderE ntaken-ntaken*)

lemma *ntake-eq-ntaken*:
assumes (*enat k*) = *m*
shows *ntake m nell* = *ntaken k nell*
using *assms* **apply** *transfer*
using *eSuc-enat* **by** *auto*

1.2.13 Concatenating a nonempty lazy list of nonempty lazy lists *nconcat*

lemma *nconcat-NNil* [*simp*]:
nconcat (*NNil nell*) = *nell*
by *transfer auto*

lemma *nconcat-NCons* [*simp*]:
nconcat (*NCons nell nells*) = *nappend nell* (*nconcat nells*)
by *transfer auto*

lemma *nconcat-def2*:
nconcat = *nellist-of-llist* ∘ *lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
by (*simp add*: *map-fun-def nconcat-def*)

lemma *not-null-lconcat*:
¬*lnull*((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
by (*simp add*: *llist-of-nellist.code*)

lemma *nconcat-def3*:
((*llist-of-nellist* ∘ *nconcat*) *nells*) =
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *llist-of-nellist* ∘ *nconcat* =
llist-of-nellist ∘ *nellist-of-llist* ∘ *lconcat* ∘ *?n2l*
by (*simp add*: *nconcat-def2 rewriteL-comp-comp*)
have *2*: ¬*lnull*((*lconcat* ∘ *?n2l*) *nells*)
using *not-null-lconcat* **by** *blast*
have *3*: (*llist-of-nellist* ∘ *nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells* =
(*lconcat* ∘ *?n2l*) *nells*
using *2 nellist-of-llist-inverse* **by** *simp*
show *?thesis*
by (*metis 1 3*)
qed

lemma *nmap-lmap-inverse*:
nmap nellist-of-llist (*nellist-of-llist* (*lmap llist-of-nellist* (*llist-of-nellist nells*))) = *nells*
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nmap nellist-of-llist* (*nellist-of-llist* (*?n2l nells*)) =
nmap nellist-of-llist (*nmap llist-of-nellist nells*)
by (*simp add*: *lmap-llist-of-nellist*)
have *2*: *nmap nellist-of-llist* (*nmap llist-of-nellist nells*) =
nmap (*nellist-of-llist*∘*llist-of-nellist*) *nells*
by (*simp add*: *nellist.map-comp*)
have *3*: (*nellist-of-llist*∘*llist-of-nellist*) = *id*
by *auto*
show *?thesis*
by (*metis 1 2 3 comp-apply nellist.map-id*)
qed

lemma *lmap-nmap-inverse*:
assumes ¬ *lnull nells*
∀ *nell*∈*lset nells*. ¬ *lnull nell*
shows (*lmap llist-of-nellist* (*llist-of-nellist* (*nmap nellist-of-llist* (*nellist-of-llist nells*)))) = *nells*
proof −
let *?l2n* = *nmap nellist-of-llist* ∘ *nellist-of-llist*
have *0*: (*nmap nellist-of-llist* (*nellist-of-llist nells*)) =
nellist-of-llist (*lmap nellist-of-llist nells*)
using *nmap-nellist-of-llist assms* **by** *simp*
have *00*: (*llist-of-nellist* (*nellist-of-llist* (*lmap nellist-of-llist nells*))) =
(*lmap nellist-of-llist nells*)
using *assms* **by** *simp*
have *1*: (*lmap llist-of-nellist* (*llist-of-nellist* (*?l2n nells*))) =
(*lmap llist-of-nellist* (((*lmap nellist-of-llist nells*))))
using *0 00* **by** *auto*
have *2*: (*lmap llist-of-nellist* (((*lmap nellist-of-llist nells*)))) =
(*lmap* (*llist-of-nellist*∘*nellist-of-llist*) *nells*)
using *llist.map-comp* **by** *blast*
have *3*: ∀ *nell* ∈ *lset nells*. (*llist-of-nellist*∘*nellist-of-llist*) *nell* = *nell*
using *assms* **by** *simp*
have *4*: (*lmap* (*llist-of-nellist*∘*nellist-of-llist*) *nells*) = *nells*
using *3* **by** *auto*
show *?thesis*
using *1 2 4 0 00* **by** *presburger*
qed

lemma *nconcat-expand*:
nconcat nells =
(*if is-NNil nells then nfirst nells else nappend* (*nfirst nells*) (*nconcat* (*ntl nells*)))
by (*metis nconcat-NCons nconcat-NNil nellist.case-eq-if nellist.collapse*(*1*) *nellist.collapse*(*2*)
nfirst-def)

lemma *lmap-llist-of-nellist-nmap*:
(*lmap* (*lmap f*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
proof −
have *5*: (*lmap* (*lmap f*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))) =
(*lmap* ((*lmap f*) ∘ *llist-of-nellist*) (*llist-of-nellist nells*))
using *llist.map-comp* **by** *blast*
have *6*: (*lmap* ((*lmap f*) ∘ *llist-of-nellist*) (*llist-of-nellist nells*)) =
(*lmap* (*llist-of-nellist* ∘ (*nmap f*)) (*llist-of-nellist nells*))
by (*simp add*: *lmap-llist-of-nellist*)
have *7*: (*lmap* (*llist-of-nellist* ∘ (*nmap f*)) (*llist-of-nellist nells*)) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
using *llist.map-comp*[*of llist-of-nellist* (*nmap f*) (*llist-of-nellist nells*)]
by *presburger*
show *?thesis*
using *5 6 7* **by** *presburger*
qed

lemma *nmap-nconcat* :
nmap f (*nconcat nells*) = *nconcat* (*nmap* (*nmap f*) *nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nmap f* (*nconcat nells*) =
nmap f ((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)
by (*simp add*: *nconcat-def2*)
have *2*: *nmap f* ((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*) =
nellist-of-llist (*lmap f* ((*lconcat* ∘ *?n2l*) *nells*))
using *nmap-nellist-of-llist*
using *not-null-lconcat* **by** *fastforce*
have *3*: (*lmap f* ((*lconcat* ∘ *?n2l*) *nells*)) =
lconcat (*lmap* (*lmap f*) (*?n2l nells*))
by (*simp add*: *lmap-lconcat*)
have *4*: (*nconcat* (*nmap* (*nmap f*) *nells*)) =
(*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (*nmap f*) *nells*)
by (*simp add*: *nconcat-def2*)
have *8*: (*lmap* (*lmap f*) (*?n2l nells*)) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
using *lmap-llist-of-nellist-nmap* **by** *auto*
show *?thesis*
using *1 2 3 4 8*
by (*metis* (*no-types*, *opaque-lifting*) *comp-eq-dest-lhs llist.map-disc-iff llist-of-nellist-inverse-b*
llist-of-nellist-not-lnull lmap-lconcat lmap-llist-of-nellist-nmap nconcat-def3
nellist-of-llist-inverse nmap-nellist-of-llist)
qed

lemma *nconcat-eq-NNil*:
nconcat nells = (*NNil x*) ⟷ *nells* = (*NNil* (*NNil x*))
by (*metis is-NNil-def is-NNil-nappend nconcat-NNil nconcat-expand*)

lemma *nhd-nconcat* [*simp*]:
⟦ ¬ *is-NNil nells*; ¬ *is-NNil* (*nhd nells*) ⟧ ⟹ *nhd* (*nconcat nells*) = *nhd* (*nhd nells*)
by (*metis nconcat-NCons nellist.collapse*(*2*) *nhd-nappend*)

lemma *ntl-nconcat* [*simp*]:
⟦ ¬ *is-NNil nells*; ¬ *is-NNil* (*nhd nells*) ⟧ ⟹
ntl (*nconcat nells*) = *nappend* (*ntl* (*nhd nells*)) (*nconcat* (*ntl nells*))
by (*metis nconcat-NCons nellist.collapse*(*2*) *ntl-nappend*)

lemma *nconcat-nappend* [*simp*]:
assumes *nfinite nells*
shows *nconcat* (*nappend nells nells1*) = *nappend* (*nconcat nells*) (*nconcat nells1*)
using *assms*
by (*induct rule*:*nfinite-induct*) (*simp-all add*: *nappend-assoc*)

lemma *nconcat-eq-NCons-conv*:
nconcat nells = *NCons x nell* ⟷

```
    nells = (NNil (NCons x nell)) ∨
    (∃ nells'. nells = (NCons (NNil x) nells') ∧ nell = nconcat nells') ∨
    (∃ nell' nells''. nells = (NCons (NCons x nell') nells'') ∧ nell = nappend nell' (nconcat nells'') )
proof (cases nells)
case (NNil nell1)
then show ?thesis by simp
next
case (NCons nell nell2)
then show ?thesis
 proof (cases is-NNil nell)
 case True
 then show ?thesis
  using NCons by simp
  (metis nappend-NCons nappend-NNil nellist.collapse(1) nellist.sel(3) nellist.sel(5))
 next
 case False
 then show ?thesis
  using NCons by simp
  (metis nappend-NCons nellist.collapse(2) nellist.distinct(1) nellist.sel(3) nellist.sel(5))
 qed
qed

lemma nlength-nconcat:
  shows nlength (nconcat nells) =
        (case nells of (NNil nell) ⇒ nlength nell |
              (NCons nell nells1) ⇒ eSuc(nlength nell) + nlength (nconcat nells1))
proof (cases nells)
case (NNil nell1)
then show ?thesis by simp
next
case (NCons nell nell2)
then show ?thesis by (simp add: eSuc-plus plus-1-eSuc(2))
qed


lemma nlength-nconcat-nfinite-conv-sum:
  assumes nfinite nells
  shows nlength (nconcat nells) =
        nlength nells + (∑ i = 0.. (the-enat (nlength nells)). nlength (nnth nells i))
using assms
proof(induct rule: nfinite-induct)
case (NNil y)
then show ?case by (simp add: zero-enat-def) (simp add: enat-0-iff(2) nnth-NNil)
next
case (NCons nell nells)
then show ?case
   proof −
    have 1: nlength (nconcat (NCons nell nells)) = 1+ nlength nell + nlength (nconcat nells)
      by simp
    have 2: nlength (nconcat nells) =
```

```
        nlength nells + (∑ i = 0.. (the-enat (nlength nells)). nlength (nnth nells i))
    using NCons.hyps(2) by blast
  have 3: nlength (NCons x nells) = 1+ nlength nells
   by (simp add: plus-1-eSuc(1))
  have 4: (∑ i = 0..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i)) =
        nlength (nnth (NCons nell nells) 0) +
        (∑ i = 1..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i))
    by (simp add: sum.atLeast-Suc-atMost)
  have 5: (∑ i = 1..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i)) =
        (∑ i = 1..(Suc (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) i))
     using NCons.hyps(1) eSuc-enat nfinite-nlength-enat by fastforce
  have 6: (∑ i = 1..(Suc (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) i)) =
        (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) (Suc i)))
    using sum.shift-bounds-cl-nat-ivl[of λ i. nlength (nnth (NCons nell nells) i) 0 1
    ( (the-enat (nlength ( nells))))]
      by simp
  have 7: (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) (Suc i))) =
        (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth ( nells) ( i)))
    by auto
  show ?thesis
  using 2 3 4 5 6 by force
 qed
qed

lemma nlength-nconcat-nfinite-conv-sum-alt:
 assumes nfinite nells
 shows  nlength nells + (∑ i = 0.. (the-enat (nlength nells)). nlength (nnth nells i)) =
       epred(∑ i = 0.. (the-enat (nlength nells)). eSuc (nlength (nnth nells i)))
using assms
proof(induct rule: nfinite-induct)
case (NNil y)
then show ?case by simp
next
case (NCons nell nells)
then show ?case
 proof −
  have 1: (∑ i = 0..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i)) =
        nlength (nnth (NCons nell nells) 0) +
        (∑ i = 1..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i))
    by (simp add: sum.atLeast-Suc-atMost)
  have 2: (∑ i = 1..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i)) =
        (∑ i = 1..(Suc (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) i))
     using NCons.hyps(1) eSuc-enat nfinite-nlength-enat by fastforce
  have 3: (∑ i = 1..(Suc (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) i)) =
        (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) (Suc i)))
    using sum.shift-bounds-cl-nat-ivl[of λ i. nlength (nnth (NCons nell nells) i) 0 1
      ( (the-enat (nlength ( nells))))]
      by simp
  have 4: (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) (Suc i))) =
        (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth ( nells) ( i)))
```

```
    by auto
  have 5: (∑ i = 0.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
=
        eSuc (nlength (nnth (NCons nell nells) 0)) +
        (∑ i = 1.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
    by (simp add: sum.atLeast-Suc-atMost)
  have 6: (∑ i = 1.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
=
        (∑ i = 1.. (Suc (the-enat (nlength ( nells)))). eSuc (nlength (nnth (NCons nell nells) i)))
    using NCons.hyps(1) eSuc-enat nfinite-nlength-enat by fastforce
  have 7: (∑ i = 1.. (Suc (the-enat (nlength ( nells)))). eSuc (nlength (nnth (NCons nell nells) i))) =
          (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth (NCons nell nells) (Suc i))))
    using sum.shift-bounds-cl-nat-ivl[of λ i. eSuc(nlength (nnth (NCons nell nells) i)) 0 1
      ( (the-enat (nlength ( nells))))]
      by simp
  have 8: (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth (NCons nell nells) (Suc i)))) =
          (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i))))
    by auto
  have 9: (∑ i = 0.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
=
        ( eSuc(nlength nell) +
              (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i)))))
    by (metis 5 6 7 8 nnth-0)
  have 10: epred(∑ i = 0.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells)
i))) =
        epred( eSuc(nlength nell) +
              (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i)))))
    using 9 by presburger
  have 11: epred( eSuc(nlength nell) +
              (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i))))) =
         eSuc(nlength nell) +
         epred(∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i))))
    by (metis add.commute eSuc-ne-0 epred-iadd1 le-add1 le-add-same-cancel1 sum.last-plus
        zero-eq-add-iff-both-eq-0)
  show ?thesis
  using 1 11 2 3 9 NCons.hyps(2) eSuc-plus by fastforce
 qed
qed


lemma nset-nconcat-nfinite:
assumes ∀ xs ∈ nset nells. nfinite xs
shows nset (nconcat nells) = (⋃ xs∈nset nells. nset xs)
proof −
 let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
 have 1: nset (nconcat nells) =
        nset (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells))
   by (simp add: nconcat-def2)
 have 2: nset (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells)) =
        lset (llist-of-nellist (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells)))
```

```
   by (metis lset-llist-of-nellist-a)
 have 3: (llist-of-nellist (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells))) =
          ( ((( lconcat ∘ ?n2l) nells)))
   using nellist-of-llist-inverse not-null-lconcat by fastforce
 have 4: lset (llist-of-nellist (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells))) =
        lset ( ((( lconcat ∘ ?n2l) nells)))
  using 3 by presburger
 have 5: ∀ nell ∈ lset (?n2l nells) . lfinite nell
    using assms nfinite-def by fastforce
 have 6: lset ( ((( lconcat ∘ ?n2l) nells))) = (⋃ nell∈ lset (?n2l nells). lset nell)
   by (simp add: 5 lset-lconcat-lfinite)
 have 7: (⋃ nell∈ lset (?n2l nells). lset nell) = ⋃ (nset ‘ nset nells)
  by simp
 show ?thesis
 by (metis 6 7 lset-llist-of-nellist-a nconcat-def3 o-apply)
qed


lemma nconcat-ntake:
shows  nconcat (ntake (enat n) nells) =
       ntake (n +(∑ i=0..n. nlength (nnth nells i))) (nconcat nells)
proof(induct n arbitrary: nells)
  case 0 thus ?case
    proof (cases nells)
    case (NNil nell)
    then show ?thesis by (simp add: zero-enat-def[symmetric]  nnth-NNil ntake-all)
    next
    case (NCons nell nell2)
    then show ?thesis by (simp add: zero-enat-def[symmetric])
       (metis dual-order.refl nlast-NNil nnth-0 ntake-all ntake-nappend1 ntaken-0 ntaken-nlast)
    qed
next
  case (Suc n)
  show ?case
  proof (cases nells)
  case (NNil nell)
  then show ?thesis
   by (metis (no-types, lifting) dual-order.trans enat-le-plus-same(1) enat-le-plus-same(2) nconcat-NNil
      nfinite-NNil nlast-NNil nlength-NNil nnth-nlast ntake-NNil ntake-all sum.atLeast0-atMost-Suc-shift
      the-enat-0)
  next
  case (NCons nell nells1)
  then show ?thesis
    proof −
     have 1: nconcat (ntake (enat (Suc n)) nells) =
             nappend nell (nconcat (ntake (enat n) nells1))
        by (metis NCons eSuc-enat nconcat-NCons ntake-Suc-NCons)
     have 2: (nconcat (ntake (enat n) nells1)) =
             ntake (enat n + (∑ i = 0..n. nlength (nnth nells1 i))) (nconcat nells1)
       using NCons Suc.hyps Suc.prems Suc-ile-eq by auto
```

```
    have 3:  nlength (nappend nell (nconcat nells1)) =
              nlength nell + 1 + nlength (nconcat nells1)
      by simp
    have 4: nappend nell (ntake (enat n + (∑ i = 0..n. nlength (nnth nells1 i))) (nconcat nells1)) =
             ntake (nlength nell + 1 + (enat n + (∑ i = 0..n. nlength (nnth nells1 i)))) (nappend nell
(nconcat nells1))
      proof (cases nlength nell = ∞)
      case True
      then show ?thesis by (metis enat-le-plus-same(2) ntake-all ntake-nappend1 plus-enat-simps(2))
      next
      case False
      then show ?thesis by (simp add: ab-semigroup-add-class.add-ac(1) enat-0-iff(1) ntake-nappend2)
      qed
    have 5: (∑ i = 0..Suc n. nlength (nnth nells i)) =
             nlength (nnth nells 0) + (∑ i = 1..Suc n. nlength (nnth nells i))
        by (simp add: sum.atLeast-Suc-atMost)
    have 6: nlength (nnth nells 0) = nlength nell
      by (simp add: NCons)
    have 7: (∑ i = 1..Suc n. nlength (nnth nells i)) =
             (∑ i = 0..n. nlength (nnth nells (Suc i)))
      using sum.shift-bounds-cl-nat-ivl[of λi . nlength (nnth nells i) 0 1 n]
      by simp
    have 8: (∑ i = 0..n. nlength (nnth nells (Suc i))) =
             (∑ i = 0..n. nlength (nnth nells1 ( i)))
      using NCons by auto
    have 9: nlength nell + 1 + (enat n + (∑ i = 0..n. nlength (nnth nells1 i))) =
             (enat (Suc n) + (∑ i = 0..Suc n. nlength (nnth nells i)))
      by (metis (no-types, lifting) 5 6 7 8 ab-semigroup-add-class.add-ac(1)
        add.left-commute eSuc-enat plus-1-eSuc(2))
    show ?thesis
    by (metis 1 2 4 9 NCons nconcat-NCons)
  qed
 qed
qed

lemma nnth-nconcat-conv:
  assumes enat n ≤ nlength (nconcat nells)
  shows ∃ m n'. nnth (nconcat nells) n = nnth (nnth nells m) n' ∧ enat n' ≤ nlength (nnth nells m) ∧
              enat m ≤ nlength nells ∧
              enat n =  (∑  i<m . eSuc(nlength (nnth nells i))) + enat n'
proof −
 let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
 have 1: ⋀nell. nlength nell = epred (llength (llist-of-nellist nell))
   unfolding nlength-def by auto
 have 2: ⋀nell j. j≤ nlength nell ⟶ nnth nell j = lnth (llist-of-nellist nell) j
   unfolding nnth-def  by auto
 have 3: nlength (nconcat nells) =
         nlength (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells))
   by (simp add: nconcat-def2)
 have 4: nlength (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells)) =
```

epred (*llength* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))))
using *nlength.rep-eq* **by** *blast*
have *5*: (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
((((*lconcat* ∘ *?n2l*) *nells*)))
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *6*: *epred* (*llength* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)))) =
epred (*llength* (((*lconcat* ∘ *?n2l*) *nells*)))
using *5* **by** *presburger*
have *7*: *enat n* < (*llength* (((*lconcat* ∘ *?n2l*) *nells*)))
by (*metis* (*no-types*, *lifting*) *3 4 5 assms co.enat.collapse iless-Suc-eq llength-eq-0*
llist-of-nellist-not-lnull)
have *8*: (((*lconcat* ∘ *?n2l*) *nells*)) =
lconcat (*?n2l nells*)
by *simp*
have *9*: ∃ *m n′*.
lnth (*lconcat* (*?n2l nells*)) *n* = *lnth* (*lnth* (*?n2l nells*) *m*) *n′* ∧
enat n′ < *llength* (*lnth* (*?n2l nells*) *m*) ∧ *enat m* < *llength* (*?n2l nells*) ∧
enat n = (∑ *i*<*m*. *llength* (*lnth* (*?n2l nells*) *i*)) + *enat n′*
using *7 lnth-lconcat-conv*[*of n* (*?n2l nells*)]
by *fastforce*
obtain *m n′* **where** *10*: *lnth* (*lconcat* (*?n2l nells*)) *n* = *lnth* (*lnth* (*?n2l nells*) *m*) *n′* ∧
enat n′ < *llength* (*lnth* (*?n2l nells*) *m*) ∧ *enat m* < *llength* (*?n2l nells*) ∧
enat n = (∑ *i*<*m*. *llength* (*lnth* (*?n2l nells*) *i*)) + *enat n′*
using *9* **by** *blast*
have *11*: *lnth* (*lconcat* (*?n2l nells*)) *n* = *nnth* (*nconcat nells*) *n*
by (*metis 2 5 8 assms nconcat-def2*)
have *12*: *lnth* (*lnth* (*?n2l nells*) *m*) *n′* = *nnth* (*nnth nells m*) *n′*
by (*metis 10 2 comp-apply co.enat.collapse iless-Suc-eq llength-eq-0 llength-lmap*
llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq)
have *13*: *llength* (*lnth* (*?n2l nells*) *m*) = *eSuc* (*nlength* (*nnth nells m*))
by (*metis 10 2 comp-apply co.enat.collapse iless-Suc-eq llength-eq-0 llength-lmap*
llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq)
have *14*: *llength* (*?n2l nells*) = *eSuc*(*nlength nells*)
by (*metis comp-apply co.enat.exhaust-sel llength-eq-0 llength-llist-of-nellist llength-lmap*
llist-of-nellist-not-lnull)
have *15*: ⋀*i*. *i*<*m* ⟹ *llength* (*lnth* (*?n2l nells*) *i*) = *eSuc*(*nlength*(*nnth nells i*))
proof −
fix *i*
assume *a*: *i*<*m*
show *llength* (*lnth* (*?n2l nells*) *i*) = *eSuc*(*nlength*(*nnth nells i*))
proof −
have *151*: (*lnth* (*?n2l nells*) *i*) = *llist-of-nellist* (*nnth nells i*)
using *a 10 14 2*
by (*metis comp-apply enat-ord-simps*(*2*) *iless-Suc-eq llength-lmap lnth-lmap order-less-trans*)
have *152*: *llength* (*llist-of-nellist* (*nnth nells i*)) = *eSuc*(*nlength*(*nnth nells i*))
using *epred-inject* **by** *force*
show *?thesis* **using** *151 152* **by** *presburger*
qed
qed
have *16*: (∑ *i*<*m*. *llength* (*lnth* (*?n2l nells*) *i*)) = (∑ *i*<*m*. *eSuc*(*nlength*(*nnth nells i*)))

by (*meson 15 lessThan-iff sum.cong*)
show *?thesis*
using *10 11 12 13 14 16* **by** (*metis iless-Suc-eq*)
qed

lemma *nnth-nconcat-ntake*:
assumes *enat w* ≤ *nlength* (*nconcat* (*ntake* (*enat n*) *nells*))
shows *nnth* (*nconcat* (*ntake* (*enat n*) *nells*)) *w* = *nnth* (*nconcat nells*) *w*
using *assms* **by** (*simp add*: *nconcat-ntake ntake-nnth*)

lemma *nfinite-nconcat* [*simp*]:
nfinite (*nconcat nells*) ⟷ *nfinite nells* ∧ (∀ *nell* ∈ *nset nells*. *nfinite nell*)
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*
thus *?rhs* (**is** *?concl nells*)
proof(*induct nconcat nells arbitrary*: *nells rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis is-NNil-imp-nfinite nconcat-eq-NNil nellist.discI*(*1*) *nellist.simps*(*20*) *singleton-iff*)
next
case (*NCons x nell*)
then show *?case*
proof (*cases nells*)
case (*NNil nell1*)
then show *?thesis* **using** *NCons.hyps* **by** *auto*
next
case (*NCons nell1 nells1*)
then show *?thesis* **using** *NCons.hyps* **by** *simp*
(*metis nconcat-NCons nellist.sel*(*5*) *nellist.set-intros*(*3*) *nfinite-NCons nfinite-nappend ntl-nappend*)
qed
qed
next
assume *?rhs*
then obtain *nfinite* (*nells*)
and ∀ *nell*∈*nset nells*. *nfinite nell* **..**
thus *?lhs*
proof(*induct nells rule*: *nfinite-induct*)
case (*NNil nell*)
then show *?case*
by *simp*
next
case (*NCons nell nells*)
then show *?case*
by (*simp add*: *nfinite-nappend*)
qed
qed

lemma *nfilter-nconcat-nfinite-help*:
assumes (∀ *nell* ∈ *nset nells*. (∃ *x* ∈ *nset nell*. *P x*))

shows (∃ *nell* ∈ *nset* (*nconcat nells*). *P nell*)
proof (*cases nells*)
case (*NNil nell*)
then show *?thesis* **using** *assms* **by** *simp*
next
case (*NCons nell nells1*)
then show *?thesis* **using** *assms*
by *simp* (*metis dual-order.trans enat-le-plus-same*(*1*) *in-nset-conv-nnth nlength-nappend nnth-nappend1*)
qed

lemma *nfilter-nconcat-nfinite*:
assumes ∀ *nell*∈*nset nells*. *nfinite nell*
∀ *nell* ∈ *nset nells*. (∃ *x* ∈ *nset nell*. *P x*)
shows *nfilter P* (*nconcat nells*) = *nconcat* (*nmap* (*nfilter P*) *nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *0*: ∃ *nell* ∈ *nset* (*nconcat nells*). *P nell*
using *assms nfilter-nconcat-nfinite-help* **by** *blast*
have *1*: *nset nells* = *lset*(*llist-of-nellist nells*)
by *simp*
have *2*: ⋀*nell*. *nell* ∈ *lset*(*llist-of-nellist nells*)⟶ *lfinite* (*llist-of-nellist nell*)
using *1 assms nfinite-def* **by** *blast*
have *3*: ⋀*nell Q*. (∃ *x*∈*nset nell*. *Q x*) ⟶
nfilter Q nell = *nellist-of-llist* (*lfilter Q* (*llist-of-nellist nell*))
unfolding *nfilter-def* **by** *simp*
have *4*: *nfilter P* (*nconcat nells*) =
nfilter P (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))
by (*simp add*: *nconcat-def2*)
have *5*: *nfilter P* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)) =
nellist-of-llist (*lfilter P* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))))
by (*metis 3 0 nconcat-def2*)
have *6*: (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
((((*lconcat* ∘ *?n2l*) *nells*)))
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *7*: (*lfilter P* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)))) =
(*lfilter P* ((((*lconcat* ∘ *?n2l*) *nells*))))
using *6* **by** *presburger*
have *8*: (((*lconcat* ∘ *?n2l*) *nells*)) = *lconcat* (*?n2l nells*)
by *simp*
have *9*: ∀ *nell*∈*lset* (*?n2l nells*). *lfinite nell*
by (*simp add*: *2*)
have *10*: (*lfilter P* ((((*lconcat* ∘ *?n2l*) *nells*)))) = *lconcat* (*lmap* (*lfilter P*) (*?n2l nells*))
by (*simp add*: *9 lfilter-lconcat-lfinite*)
have *11*: *nconcat* (*nmap* (*nfilter P*) *nells*) =
(((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (*nfilter P*) *nells*)))
by (*simp add*: *nconcat-def2*)
have *12*: ⋀*nell f*. *nmap f nell* = *nellist-of-llist* (*lmap f* (*llist-of-nellist nell*))
by (*metis llist-of-nellist-inverse-b llist-of-nellist-not-lnull nmap-nellist-of-llist*)
have *13*: (*nmap* (*nfilter P*) *nells*) = *nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*))
using *12* **by** *blast*

have *14*: *nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*)) =
nellist-of-llist (*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*))
by (*metis* (*mono-tags*, *lifting*) *1 3 assms*(*2*) *llist.map-cong*)
have *15*: *lmap* (*lfilter P*) (*lmap llist-of-nellist* (*llist-of-nellist nells*)) =
lmap ((*lfilter P*)∘ *llist-of-nellist*) (*llist-of-nellist nells*)
using *llist.map-comp* **by** *blast*
have *16*: ((*?n2l* (*nmap* (*nfilter P*) *nells*))) =
((*?n2l* (*nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*)))))
using *13* **by** *presburger*
have *17*: ((*?n2l* (*nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*))))) =
((*?n2l* (*nellist-of-llist*
(*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*)))))
using *14* **by** *presburger*
have *18*: ((*?n2l* (*nellist-of-llist*
(*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*))))) =
((*lmap llist-of-nellist*
(((*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*))))))
by *simp*
have *19*: ((*lmap llist-of-nellist*
(((*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*)))))) =
((*lmap* (*llist-of-nellist*∘(λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*)))) (*llist-of-nellist nells*)))
using *llist.map-comp* **by** *blast*
have *20*: ⋀ *nell*. *nell* ∈ *lset*(*llist-of-nellist nells*) ⟶ ¬*lnull* (*lfilter P* (*llist-of-nellist nell*))
by (*simp add*: *assms*(*2*))
have *21*: ((*lmap* (*llist-of-nellist*∘(λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*)))) (*llist-of-nellist nells*)))
=
((*lmap* ((λ*ys*. (*lfilter P* (*llist-of-nellist ys*)))) (*llist-of-nellist nells*)))
by (*metis* (*no-types*, *lifting*) *20 comp-def llist.map-cong0 nellist-of-llist-inverse*)
have *22*: ((*lmap llist-of-nellist* (*llist-of-nellist* (*nmap* (*nfilter P*) *nells*)))) =
(*lmap* (*lfilter P*) (((((*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*))))
using *13 14 15 19 21* **by** *force*
have *23*: *nellist-of-llist* (*lconcat* (*lmap* (*lfilter P*) (*?n2l nells*))) =
nconcat (*nmap* (*nfilter P*) *nells*)
by (*simp add*: *22 nconcat-def2*)
show *?thesis*
by (*metis 10 23 5 6 nconcat-def2*)
qed

lemma *nconcat-nmap-singleton* [*simp*]:
nconcat (*nmap* (λ*x*. *NNil* (*f x*)) *nell*) = *nmap f nell*
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nconcat* (*nmap* (λ*x*. *NNil* (*f x*)) *nell*) =
(((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (λ*x*. *NNil* (*f x*)) *nell*)))
by (*simp add*: *nconcat-def2*)
have *2*: (*nmap* (λ*x*. *NNil* (*f x*)) *nell*) =
nellist-of-llist (*lmap* (λ*x*. *NNil* (*f x*)) (*llist-of-nellist nell*))
by (*simp add*: *lmap-llist-of-nellist*)
have *3*: *nellist-of-llist* (*lmap* (λ*x*. *NNil* (*f x*)) (*llist-of-nellist nell*)) =
nellist-of-llist (*lmap* (λ*x*. *nellist-of-llist* (*LCons* (*f x*) *LNil*)) (*llist-of-nellist nell*))

by *force*
have *4*: *nmap f nell* = *nellist-of-llist* (*lmap f* (*llist-of-nellist nell*))
by (*simp add*: *lmap-llist-of-nellist*)
have *5*: *lconcat* (*?n2l* (*nellist-of-llist*
(*lmap* (λ*x*. *nellist-of-llist* (*LCons* (*f x*) *LNil*)) (*llist-of-nellist nell*)))) =
lconcat (*lmap llist-of-nellist* (*lmap* (λ*z*. *NNil* (*f z*)) (*llist-of-nellist nell*)))
by *simp*
have *6*: (*lmap llist-of-nellist* (*lmap* (λ*z*. *NNil* (*f z*)) (*llist-of-nellist nell*))) =
(*lmap* (*llist-of-nellist*∘(λ*z*. *NNil* (*f z*))) (*llist-of-nellist nell*))
using *llist.map-comp* **by** *metis*
have *7*: (*lmap* (*llist-of-nellist*∘(λ*z*. *NNil* (*f z*))) (*llist-of-nellist nell*)) =
(*lmap* (λ*z*. *LCons* (*f z*) *LNil*) (*llist-of-nellist nell*))
by *auto*
have *8*: *lconcat* (*?n2l* (*nellist-of-llist*
(*lmap* (λ*x*. *nellist-of-llist* (*LCons* (*f x*) *LNil*)) (*llist-of-nellist nell*))))
= (*lmap f* (*llist-of-nellist nell*))
using *6* **by** *force*
have *9*: (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (λ*x*. *NNil* (*f x*)) *nell*))) =
nellist-of-llist (*lmap f* (*llist-of-nellist nell*))
using *2 8* **by** *auto*
show *?thesis*
using *1 4 9* **by** *presburger*
qed

lemma *nset-nconcat-subset*:
nset (*nconcat nells*) ⊆ (⋃ *nell*∈*nset nells*. *nset nell*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nset* (*nconcat nells*) =
nset (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))
by (*simp add*: *nconcat-def2*)
have *2*: *nset* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)) =
lset (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)))
by (*metis lset-llist-of-nellist-a*)
have *3*: (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
((((*lconcat* ∘ *?n2l*) *nells*)))
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *4*: *lset* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
lset ((((*lconcat* ∘ *?n2l*) *nells*)))
using *3* **by** *presburger*
have *5*: *lset* ((((*lconcat* ∘ *?n2l*) *nells*))) ⊆
(⋃ *nell*∈*lset* (*?n2l nells*). *lset nell*)
using *lset-lconcat-subset* **by** *fastforce*
have *6*: (⋃ *nell*∈*lset* (*?n2l nells*). *lset nell*) =
⋃ (*nset* ‘ *nset nells*)
by *simp*
show *?thesis*
by (*metis 1 2 3 5 6*)
qed

lemma *ndistinct-nconcat*:
assumes *ndistinct nells*
$\bigwedge$*nell. nell* $\in$ *nset nells* $\Longrightarrow$ *ndistinct nell*
$\bigwedge$*nell nell1.* $[\![$ *nell* $\in$ *nset nells*; *nell1* $\in$ *nset nells*; *nell* $\neq$ *nell1* $]\!] \Longrightarrow$ *nset nell* $\cap$ *nset nell1* $= \{\}$
shows *ndistinct* (*nconcat nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)
have *1*: *ldistinct* (*llist-of-nellist nells*)
using *assms*(*1*) *ndistinct.rep-eq* **by** *auto*
have *2*: $\bigwedge$*nell. nell* $\in$ *lset* (*llist-of-nellist nells*) $\Longrightarrow$ *ldistinct* (*llist-of-nellist nell*)
using *assms*(*2*) *ndistinct.rep-eq* **by** *auto*
have *3*: $\bigwedge$*nell nell1.* $[\![$ *nell* $\in$ *lset* (*llist-of-nellist nells*); *nell1* $\in$ *lset* (*llist-of-nellist nells*); *nell* $\neq$ *nell1* $]\!]$
$\Longrightarrow$ *lset* (*llist-of-nellist nell*) $\cap$ *lset* (*llist-of-nellist nell1*) $= \{\}$
using *assms*(*3*) **by** *simp*
have *4*: *ndistinct* (*nconcat nells*) =
ndistinct (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*))
by (*simp add*: *nconcat-def2*)
have *5*: *ndistinct* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*)) =
ldistinct (*llist-of-nellist* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*)))
using *ndistinct.rep-eq* **by** *blast*
have *6*: (*llist-of-nellist* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*))) =
((((*lconcat* $\circ$ *?n2l*) *nells*)))
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *7*: *ldistinct* (*llist-of-nellist* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*))) =
ldistinct ((((*lconcat* $\circ$ *?n2l*) *nells*)))
using *6* **by** *presburger*
have *8*: *ldistinct* (*?n2l nells*)
by (*metis 1 bi-unique-cr-nellist-help comp-eq-dest-lhs inj-on-def ldistinct-lmap*)
have *9*: $\bigwedge$*nell. nell* $\in$ *lset* (*?n2l nells*) $\Longrightarrow$ *ldistinct nell*
using *2* **by** *auto*
have *10*: $\bigwedge$*nell nell1.* $[\![$ *nell* $\in$ *lset* (*?n2l nells*);
nell1 $\in$ *lset* (*?n2l nells*); *nell* $\neq$ *nell1*$]\!] \Longrightarrow$
lset nell $\cap$ *lset nell1* $= \{\}$
by (*metis* (*no-types*, *lifting*) *assms*(*3*) *comp-def imageE lset-llist-of-nellist-a lset-lmap*)
have *11*: *ldistinct* (*lconcat* (*?n2l nells*))
using *10 8 9 ldistinct-lconcat* **by** *blast*
show *?thesis*
using *11 4 5 6* **by** *fastforce*
qed

1.2.14 *nellist-all2*

lemmas *nellist-all2-NNil* = *nellist.rel-inject*(*1*)
lemmas *nellist-all2-NCons* = *nellist.rel-inject*(*2*)

lemma *nellist-all2-NNil1*:
nellist-all2 Q (*NNil b*) *nell* $\longleftrightarrow$ ($\exists\, b'$. *nell* = *NNil b'* $\wedge$ *Q b b'*)
using *nellist.rel-cases* **by** *fastforce*

lemma *nellist-all2-NNil2*:

nellist-all2 Q nell (*NNil b'*) ⟷ (∃ *b*. *nell* = *NNil b* ∧ *Q b b'*)
using *nellist.rel-sel*
by (*metis is-NNil-def nellist-all2-NNil*)

lemma *nellist-all2-NCons1*:
nellist-all2 P (*NCons x nell*) *nell'* ⟷
(∃ *x' nell''*. *nell'* = *NCons x' nell''* ∧ *P x x'* ∧ *nellist-all2 P nell nell''*)
using *nellist.rel-sel*
by (*metis nellist.collapse*(*2*) *nellist.disc*(*2*) *nellist.sel*(*3*) *nellist.sel*(*5*))

lemma *nellist-all2-NCons2*:
nellist-all2 P nell' (*NCons x nell*) ⟷
(∃ *x' nell''*. *nell'* = *NCons x' nell''* ∧ *P x' x* ∧ *nellist-all2 P nell'' nell*)
by (*metis nellist.collapse*(*2*) *nellist.disc*(*2*) *nellist.rel-sel nellist.sel*(*3*) *nellist.sel*(*5*))

lemma *nellist-all2-coinduct* [*consumes 1*, *case-names ilist-all2*,
case-conclusion nellist-all2 is-NNil NNil NCons,
coinduct pred: *nellist-all2*]:
assumes *X nellx nelly*
and ⋀*nellix nelliy*.
X nellix nelliy ⟹
(*is-NNil nellix* = *is-NNil nelliy*) ∧
(*is-NNil nellix* ⟶ *is-NNil nelliy* ⟶ *P* (*nlast nellix*) (*nlast nelliy*)) ∧
(¬ *is-NNil nellix* ⟶ ¬ *is-NNil nelliy* ⟶ *P* (*nhd nellix*) (*nhd nelliy*) ∧
(*X* (*ntl nellix*) (*ntl nelliy*) ∨ *nellist-all2 P* (*ntl nellix*) (*ntl nelliy*)))
shows *nellist-all2 P nellx nelly*
using *assms*
nellist.rel-coinduct[*of* (λ *nelx nely*. *X nelx nely* ∨ *nellist-all2 P nelx nely*) *nellx nelly P*]
by (*metis nellist.rel-sel*)

lemma *nellist-all2-cases*[*consumes 1*, *case-names NNil NCons*, *cases pred*]:
assumes *nellist-all2 P nellx nelly*
obtains (*NNil*) *b b'* **where** *nellx* = *NNil b nelly* = *NNil b' P b b'*
| (*NCons*) *x nellx' y nelly'*
where *nellx* = *NCons x nellx'* **and** *nelly* = *NCons y nelly'*
and *P x y* **and** *nellist-all2 P nellx' nelly'*
using *assms*
using *nellist.rel-cases* **by** *blast*

lemma *nellist-all2-nmap*:
nellist-all2 P (*nmap f nellx*) *nelly* ⟷ *nellist-all2* (λ*x y*. *P* (*f x*) *y*) *nellx nelly*
using *nellist.rel-map*(*1*) **by** *blast*

lemma *nellist-all2-nmap2*:
nellist-all2 P nellx (*nmap f nelly*) ⟷ *nellist-all2* (λ*x y*. *P x* (*f y*)) *nellx nelly*
using *nellist.rel-map*(*2*) **by** *blast*

lemma *nellist-all2-mono*:
⟦ *nellist-all2 P nellx nelly*; ⋀*x y*. *P x y* ⟹ *P' x y* ⟧
⟹ *nellist-all2 P' nellx nelly*

using *nellist.rel-mono-strong* **by** *blast*

lemma *nellist-all2-nlengthD*:
nellist-all2 P nellx nelly $\Longrightarrow$ *nlength nellx* = *nlength nelly*
by(*transfer*)(*auto dest*: *llist-all2-llengthD*)

lemma *nellist-all2-nfiniteD*:
nellist-all2 P nellx nelly $\Longrightarrow$ *nfinite nellx* = *nfinite nelly*
by *transfer*
(*auto dest*: *llist-all2-lfiniteD*)

lemma *nellist-all2-nfinite1-nlastD*:
$[\![$ *nellist-all2 P nellx nelly*; *nfinite nellx* $]\!]$ $\Longrightarrow$ *P* (*nlast nellx*) (*nlast nelly*)
by (*frule nellist-all2-nfiniteD*)
(*transfer*,
auto simp add: *llist-all2-conv-all-lnth*,
metis Suc-ile-eq eSuc-enat lfinite.simps lfinite-llength-enat llast-conv-lnth llength-LCons
llist.discI(*1*) *order-refl*)

lemma *nellist-all2-nfinite2-nlastD*:
$[\![$ *nellist-all2 P nellx nelly*; *nfinite nelly* $]\!]$ $\Longrightarrow$ *P* (*nlast nellx*) (*nlast nelly*)
by(*metis nellist-all2-nfinite1-nlastD nellist-all2-nfiniteD*)

lemma *nellist-all2D-llist-all2-llist-of-nellist*:
nellist-all2 P nellx nelly $\Longrightarrow$ *llist-all2 P* (*llist-of-nellist nellx*) (*llist-of-nellist nelly*)
by *transfer*
(*simp add*: *nellist-all2-help-b*)

lemma *nellist-all2-is-NNilD*:
nellist-all2 P nellx nelly $\Longrightarrow$ *is-NNil nellx* $\longleftrightarrow$ *is-NNil nelly*
by (*cases nellx*) (*auto simp add*: *nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-nhdD*:
$[\![$ *nellist-all2 P nellx nelly*; $\neg$ *is-NNil nellx* $\vee$ $\neg$ *is-NNil nelly* $]\!]$ $\Longrightarrow$ *P* (*nhd nellx*) (*nhd nelly*)
by (*cases nellx*) (*auto simp add*: *nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-ntlI*:
$[\![$ *nellist-all2 P nellx nelly*; $\neg$ *is-NNil nellx* $\vee$ $\neg$ *is-NNil nelly* $]\!]$ $\Longrightarrow$
nellist-all2 P (*ntl nellx*) (*ntl nelly*)
by (*cases nellx*) (*auto simp add*: *nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-refl*:
nellist-all2 P nell nell $\longleftrightarrow$
($\forall$ *x* $\in$ *nset nell*. *P x x*) $\wedge$ (*nfinite nell* $\longrightarrow$ *P* (*nlast nell*) (*nlast nell*))
by *transfer*
(*auto*, *metis in-lset-lappend-iff lappend-lbutlast-llast-id-lfinite lfinite-lappend*
llist.set-intros(*1*))

lemma *nellist-all2-reflI*:
$[\![$ $\bigwedge$*x*. *x* $\in$ *nset nell* $\Longrightarrow$ *P x x*; *nfinite nell* $\Longrightarrow$ *P* (*nlast nell*) (*nlast nell*) $]\!]$

$\Longrightarrow$ *nellist-all2 P nell nell*
by(*simp add*: *nellist-all2-refl*)

lemma *nellist-all2-conv-all-nnth-help1*:
$\neg$ *lnull nellx* $\Longrightarrow$ $\neg$ *lnull nelly* $\Longrightarrow$ *lfinite nelly* $\Longrightarrow$ *llist-all2 P nellx nelly* $\Longrightarrow$
P (*llast nellx*) (*llast nelly*)
proof −
assume *a1*: *lfinite nelly*
assume *a2*: *llist-all2 P nellx nelly*
assume *a3*: $\neg$ *lnull nellx*
assume *a4*: $\neg$ *lnull nelly*
have *f5*: *lfinite* (*lappend* (*lbutlast nellx*) (*LCons* (*llast nellx*) *LNil*))
using *a3 a2 a1* **by** (*simp add*: *llist-all2-lfiniteD*)
have *f6*: *llength* (*ltl nellx*) = *epred* (*llength nelly*)
using *a2* **by** (*metis* (*no-types*) *epred-llength llist-all2-llengthD*)
have *f7*: *lappend* (*lbutlast nelly*) (*LCons* (*llast nelly*) *LNil*) = *nelly*
using *a4* **by** (*meson lappend-lbutlast-llast-id*)
have *llength* (*lbutlast nellx*) = *llength* (*ltl nellx*)
using *epred-llength* **by** *auto*
then show *?thesis*
using *f7 f6 f5 a3 a2 a1*
by (*metis* (*no-types*) *lappend-eq-lappend-conv lappend-lbutlast-llast-id lappend-ltake-ldrop*
lbutlast-conv-ltake lfinite-lappend lhd-LCons llist.disc(*2*) *llist-all2-lappend1D*(*2*)
llist-all2-lhdD2)
qed

lemma *nellist-all2-conv-all-nnth*:
nellist-all2 P nellx nelly $\longleftrightarrow$
nlength nellx = *nlength nelly* $\wedge$
($\forall$ *n*. *enat n* $\leq$ *nlength nellx* $\longrightarrow$ *P* (*nnth nellx n*) (*nnth nelly n*))
by *transfer*
(*auto simp add*: *llist-all2-llengthD*,
metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-lnthD2,
metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-conv-all-lnth)

lemma *nellist-all2-True* [*simp*]:
nellist-all2 (λ- -. *True*) *nellx nelly* $\longleftrightarrow$ *nlength nellx* = *nlength nelly*
by(*simp add*: *nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nnthD*:
⟦ *nellist-all2 P nellx nelly*; *enat n* $\leq$ *nlength nellx* ⟧ $\Longrightarrow$ *P* (*nnth nellx n*) (*nnth nelly n*)
by (*simp add*: *nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nnthD2*:
⟦ *nellist-all2 P nellx nelly*; *enat n* $\leq$ *nlength nelly* ⟧ $\Longrightarrow$ *P* (*nnth nellx n*) (*nnth nelly n*)
by (*simp add*: *nellist-all2-conv-all-nnth*)

lemmas *nellist-all2-eq* = *nellist.rel-eq*

lemma *nmap-eq-nmap-conv-nellist-all2*:
nmap f nellx = nmap f′ nelly ⟷
nellist-all2 (λx y. f x = f′ y) nellx nelly
by *transfer*
(clarsimp simp add: lmap-eq-lmap-conv-llist-all2)

lemma *nellist-all2-trans*:
⟦ nellist-all2 P nellx nelly; nellist-all2 P nelly nellz; transp P ⟧
⟹ nellist-all2 P nellx nellz
by *transfer (auto elim: llist-all2-trans dest: llist-all2-lfiniteD transpD)*

lemma *nellist-all2-nappendI*:
⟦ nellist-all2 P nellx nelly;
⟦ nfinite nellx; nfinite nelly; P (nlast nellx) (nlast nelly) ⟧
⟹ nellist-all2 P nellx′ nelly′ ⟧
⟹ nellist-all2 P (nappend nellx nellx′) (nappend nelly nelly′)
by *transfer*
(auto simp add: lnull-def lappend-eq-lappend-conv nellist-all2-conv-all-nnth-help1
intro: llist-all2-lappendI)

lemma *nested-nellist-all2-nested-llist-all2*:
nellist-all2 (nellist-all2 A) nells nells1 =
llist-all2 (llist-all2 A) (lmap llist-of-nellist (llist-of-nellist nells))
(lmap llist-of-nellist (llist-of-nellist nells1))
proof −
have *1*: *nellist-all2 (nellist-all2 A) nells nells1 =*
llist-all2 (nellist-all2 A) (llist-of-nellist nells) (llist-of-nellist nells1)
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*
have *2*: *(λ xx yy. nellist-all2 A xx yy) =*
(λ xx yy. llist-all2 A (llist-of-nellist xx) (llist-of-nellist yy))
by *(meson nellist-all2D-llist-all2-llist-of-nellist nellist-all2-help-a)*
have *3*: *llist-all2 (nellist-all2 A) (llist-of-nellist nells) (llist-of-nellist nells1) =*
llist-all2 (λ xx yy. llist-all2 A (llist-of-nellist xx) (llist-of-nellist yy))
(llist-of-nellist nells) (llist-of-nellist nells1)
using *2* **by** *auto*
have *6*: *llist-all2 (λ xx yy. llist-all2 A (llist-of-nellist xx) (llist-of-nellist yy))*
(llist-of-nellist nells) (llist-of-nellist nells1) =
llist-all2 (llist-all2 A) (lmap llist-of-nellist (llist-of-nellist nells))
(lmap llist-of-nellist (llist-of-nellist nells1))
using *llist-all2-lmap1[of (llist-all2 A) llist-of-nellist (llist-of-nellist nells)*
(lmap llist-of-nellist (llist-of-nellist nells1))]
llist-all2-lmap2[of (llist-all2 A) - llist-of-nellist (llist-of-nellist nells1)]
by *(simp add: llist-all2-conv-all-lnth)*
show *?thesis*
using *1 3 6* **by** *blast*
qed

lemma *nellist-all2-nconcatI*:
assumes *nellist-all2 (nellist-all2 A) nells nells1*
shows *nellist-all2 A (nconcat nells) (nconcat nells1)*

proof −
have *3*: *nellist-all2 A* (*nconcat nells*) (*nconcat nells1*) =
llist-all2 A (*llist-of-nellist* (*nconcat nells*)) (*llist-of-nellist* (*nconcat nells1*))
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*
have *4*: (*llist-of-nellist* (*nconcat nells*)) =
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
using *nconcat-def3* **by** *auto*
have *5*: (*llist-of-nellist* (*nconcat nells1*)) =
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells1*)
using *nconcat-def3* **by** *auto*
have *7*: *llist-all2* (*llist-all2 A*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))
(*lmap llist-of-nellist* (*llist-of-nellist nells1*)) ⟹
llist-all2 A ((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells1*)
using *llist-all2-lconcatI*[*of A* (*lmap llist-of-nellist* (*llist-of-nellist nells*))
(*lmap llist-of-nellist* (*llist-of-nellist nells1*))]
by *simp*
have *8*:
llist-all2 A ((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells1*)
= *nellist-all2 A* (*nconcat nells*) (*nconcat nells1*)
using *3 4 5* **by** *presburger*
have *9*: *nellist-all2* (*nellist-all2 A*) *nells nells1* =
llist-all2 (*llist-all2 A*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))
(*lmap llist-of-nellist* (*llist-of-nellist nells1*))
using *nested-nellist-all2-nested-llist-all2* **by** *blast*
show *?thesis* **using** *assms 7 8 9*
by *blast*
qed

lemma *nlength-nconcat-eqI*:
fixes *nells* :: *'a nellist nellist* **and** *nells1* :: *'b nellist nellist*
assumes *nellist-all2* (*λxs ys. nlength xs = nlength ys*) *nells nells1*
shows *nlength* (*nconcat nells*) = *nlength* (*nconcat nells1*)
proof −
have *nellist-all2* (*nellist-all2* (*λa b. True*)) *nells nells1*
using *assms nellist.rel-mono-strong nellist-all2-True* **by** *blast*
then show *?thesis* **using** *nellist-all2-nconcatI nellist-all2-nlengthD* **by** *blast*
qed

lemma *llist-all2-nellist-of-llistI*:
nellist-all2 A nellx nelly ⟹
llist-all2 A (*lbutlast*(*llist-of-nellist nellx*)) (*lbutlast*(*llist-of-nellist nelly*))
proof (*coinduction arbitrary*: *nellx nelly*)
case *LNil*
then show *?case*
by (*metis lbutlast.disc-iff*(*1*) *llist.disc*(*1*) *llist-of-nellist-inverse-a ltl-llist-of-nellist1*
nellist-all2-is-NNilD nellist-of-llist-a.disc(*1*))

next
case (*LCons nell1 nell2*)
then show *?case*
proof −
assume *a0*: *nellist-all2 A nell1 nell2*
assume *a1*: ¬ *lnull* (*lbutlast* (*llist-of-nellist nell1*))
assume *a2*: ¬ *lnull* (*lbutlast* (*llist-of-nellist nell2*))
have *1*: *A* (*lhd* (*lbutlast* (*llist-of-nellist nell1*))) (*lhd* (*lbutlast* (*llist-of-nellist nell2*)))
using *a0 a1 a2*
by (*metis lbutlast.disc*(*1*) *llist.disc*(*1*) *llist-of-nellist-inverse-a ltl-llist-of-nellist1*
nellist-all2-nhdD nhd-nellist-of-llist-a)
have *2*: ((∃ *nellx nelly*.
ltl (*lbutlast* (*llist-of-nellist nell1*)) = *lbutlast* (*llist-of-nellist nellx*) ∧
ltl (*lbutlast* (*llist-of-nellist nell2*)) = *lbutlast* (*llist-of-nellist nelly*) ∧
nellist-all2 A nellx nelly) ∨
llist-all2 A (*ltl* (*lbutlast* (*llist-of-nellist nell1*))) (*ltl* (*lbutlast* (*llist-of-nellist nell2*))))
by (*metis a0 a1 lbutlast.ctr*(*1*) *lbutlast-ltl llist.discI*(*1*) *ltl-llist-of-nellist ltl-llist-of-nellist1*
nellist.rel-sel)
show *?thesis* **using** *1 2* **by** *blast*
qed
qed

lemma *nellist-all2-nellist-of-llist-a* [*simp*]:
nellist-all2 A (*nellist-of-llist-a b llx*) (*nellist-of-llist-a c lly*) ⟷
llist-all2 A llx lly ∧ (*lfinite llx* ⟶ *A b c*)
proof (*cases lfinite llx*)
case *True*
then show *?thesis*
using *llist-all2-nellist-of-llistI*
by (*auto simp add*: *llist-all2-lappendI nellist-all2-help-a*, *fastforce*,
metis lbutlast-lfinite lbutlast-snoc llist-all2-lfiniteD nellist-all2-nfinite1-nlastD
nellist-of-llist-a-inverse nfinite-def nlast-nellist-of-llist-a-lfinite)
next
case *False*
then show *?thesis*
by (*metis lbutlast-snoc llist-all2-lappendI llist-all2-nellist-of-llistI nellist-all2-help-a*
nellist-of-llist-a-inverse)
qed

1.2.15 From a nonempty lazy list to a lazy list *llist-of-nellist*

lemma *llist-of-nellist-nmap* [*simp*]:
llist-of-nellist (*nmap f nell*) = *lmap f* (*llist-of-nellist nell*)
by (*simp add*: *lmap-llist-of-nellist*)

lemma *llist-of-nellist-nappend*:
llist-of-nellist (*nappend nellx nelly*) = *lappend* (*llist-of-nellist nellx*) (*llist-of-nellist nelly*)
by (*transfer*) *auto*

lemma *llist-of-nellist-lappendn* [*simp*]:

llist-of-nellist (*lappendn ll nell*) = *lappend ll* (*llist-of-nellist nell*)
by *transfer auto*

lemma *llist-of-nellist-nconcat* [*simp*]:
llist-of-nellist (*nconcat nell*) = *lconcat* ((*lmap llist-of-nellist* ∘ *llist-of-nellist*) *nell*)
using *nconcat-def3* **by** *fastforce*

lemma *llist-of-nellist-nfilter* [*simp*]:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *llist-of-nellist* (*nfilter P nell*) = *lfilter P* (*llist-of-nellist nell*)
using *assms*
by *transfer auto*

1.2.16 *ndropn*

lemma *ndropn-0* [*simp*, *code*, *nitpick-simp*]:
ndropn 0 nell = *nell*
using *zero-enat-def* **by** *transfer auto*

lemma *ndropn-NNil* [*simp*, *code*]:
ndropn n (*NNil b*) = (*NNil b*)
by *transfer auto*

lemma *ndropn-Suc-NCons* [*simp*, *code*]:
ndropn (*Suc n*) (*NCons x nell*) = *ndropn n nell*
proof (*cases nfinite nell*)
case *True*
then show *?thesis*
by *transfer*
(*auto simp add*: *min-def not-lnull-conv Suc-ile-eq llength-eq-infty-conv-lfinite the-enat-eSuc* ,
metis Extended-Nat.eSuc-mono eSuc-enat iless-Suc-eq leD,
metis antisym eSuc-enat enat-the-enat ile-eSuc llength-eq-infty-conv-lfinite n-not-Suc-n
the-enat.simps)
next
case *False*
then show *?thesis*
by *transfer*
(*simp*,
metis co.enat.sel(*2*) *eSuc-infinity infinity-ileE ldropn-Suc-LCons llength-eq-infty-conv-lfinite*
min.cobounded1 min-def the-enat.simps)
qed

lemma *ndropn-Suc* [*nitpick-simp*]:
ndropn (*Suc n*) *nell* = (*case nell of NNil b* ⇒ *NNil b* | *NCons x nell′* ⇒ *ndropn n nell′*)
by(*cases nell*) *simp-all*

lemma *ltl-power-NNil-help*:
((λ*ll*. *if* ∃ *b*. *ll* = *LCons b LNil then ll else ltl ll*) ^^ *n*) (*LCons b LNil*) = *LCons b LNil*
by (*induction n*) *simp-all*

lemma *ntl-power-NNil*:
(*ntl* ^^ *n*) (*NNil b*) = (*NNil b*)
by *transfer* (*auto simp add*: *lnull-def ltl-power-NNil-help*)

lemma *ntl-power-NCons*:
ntl ((*ntl* ^^ *n*) (*NCons x nell*)) = (*ntl* ^^ *n*) *nell*
by (*induction n*) (*transfer*, *auto*)

lemma *ntl-power-Suc* [*simp*]:
(*ntl* ^^ (*Suc n*)) *nell* = (*case nell of NNil b* ⇒ *NNil b* | *NCons x nell'* ⇒ (*ntl* ^^ *n*) *nell'*)
by (*cases nell*)
(*simp-all add*: *ntl-power-NNil ntl-power-NCons*)

lemma *llist-of-nellist-ndropn* [*simp*]:
llist-of-nellist (*ndropn n nell*) =
ldropn (*the-enat* (*min* (*enat n*) ((*epred*(*llength*((*llist-of-nellist nell*)))))))
(*llist-of-nellist nell*)
by *transfer auto*

lemma *ndropn-Suc-conv-ndropn*:
enat n < *nlength nell* ⟹ *NCons* (*nnth nell n*) (*ndropn* (*Suc n*) *nell*) = *ndropn n nell*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **using** *0.prems* **by** *auto*
next
case (*NCons x nell1*)
then show *?thesis* **by** *simp*
qed
next
case (*Suc n*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **using** *Suc.prems* **by** *auto*
next
case (*NCons x nell1*)
then show *?thesis* **using** *Suc*
by (*metis One-nat-def add.commute add-left-mono gen-nlength-code*(*2*) *gen-nlength-def leD*
ndropn-Suc-NCons nlength-code nnth-Suc-NCons not-le-imp-less of-nat-Suc of-nat-eq-enat
one-enat-def plus-1-eq-Suc)
qed
qed

lemma *ndropn-nlength* [*simp*]:
nlength (*ndropn n nell*) = *nlength nell* − *enat n*
proof (*induct n arbitrary*: *nell*)

case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x nell1*)
then show *?thesis* **using** *Suc*
by (*metis eSuc-enat eSuc-minus-eSuc ndropn-Suc-NCons nlength-NCons*)
qed
qed

lemma *ndropn-nnth* [*simp*]:
nnth (*ndropn n nell*) *m* = *nnth nell* (*n*+*m*)
proof (*induct n arbitrary*: *m nell*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** (*simp add*: *nnth-NNil*)
next
case (*NCons x nell1*)
then show *?thesis* **by** (*simp add*: *Suc.hyps*)
qed
qed

lemma *ndropn-nnth-a*:
assumes *nlength nell* ≤ *enat*(*n*+*m*)
shows *nnth* (*ndropn n nell*) *m* = (*nlast nell*)
proof −
have *1*: *nfinite nell*
using *assms enat-ile nfinite-conv-nlength-enat* **by** *auto*
have *2*: *nfinite nell* ⟹ *nlength nell* ≤ *enat*(*n*+*m*) ⟹ *nnth* (*ndropn n nell*) *m* = (*nlast nell*)
proof (*induct arbitrary*: *n m rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*simp add*: *nnth-NNil*)
next
case (*NCons x nell*)
then show *?case*
proof (*cases n*)
case *0*
then show *?thesis*
by (*metis NCons*(*2*) *NCons*(*3*) *Suc-ile-eq Suc-pred add-cancel-right-left enat-le-plus-same*(*1*)
gen-nlength-def iless-Suc-eq leD le-add1 ndropn-0 nellist.sel(*2*) *nlast0-nlast nlength-NCons*

nlength-code nnth-Suc-NCons not-le-imp-less order.not-eq-order-implies-strict)
next
case (*Suc nat*)
then show *?thesis*
by (*metis NCons*(*2*) *NCons*(*3*) *add-Suc eSuc-enat epred-eSuc epred-le-epredI ndropn-Suc-NCons nlast-NCons nlength-NCons*)
qed
qed
show *?thesis* **using** *1 2 assms* **by** *auto*
qed

lemma *ndropn-ntl* :
ndropn n nell = (*ntl* ^^ *n*) *nell*
proof (*induction n arbitrary*: *nell*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
by (*simp add*: *ntl-power-NNil*)
next
case (*NCons x nell*)
then show *?thesis*
by (*metis Suc.IH funpow-Suc-right ndropn-Suc-NCons nellist.sel*(*5*) *o-apply*)
qed
qed

lemma *ndropn-is-NNil*:
is-NNil nell ⟹ *ndropn n nell* = *nell*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by** (*simp add*: *ndropn-Suc nellist.case-eq-if*)
qed

lemma *is-NNil-ndropn*:
is-NNil(*ndropn n nell*) ⟷ *nlength nell* ≤ (*enat n*)
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x nell*)

then show *?thesis* **using** *zero-enat-def* **by** *auto*
qed
next
case (*Suc n*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x nell*)
then show *?thesis*
by (*metis One-nat-def Suc.hyps add.commute add-left-mono co.enat.sel(2) eSuc-enat epred-le-epredI gen-nlength-code(2) gen-nlength-def ndropn-Suc-NCons nlength-NCons nlength-code of-nat-Suc of-nat-eq-enat one-enat-def plus-1-eq-Suc*)
qed
qed

lemma *ndropn-eq-NNil*:
ndropn n nell = (*NNil b*) ⟷ *nnth nell* (*the-enat*(*nlength nell*)) = *b* ∧ *nlength nell* ≤ (*enat n*)
proof −
have *1*: *nfinite nell* ⟹ *ndropn n nell* = *NNil b* ⟹ *nnth nell* (*the-enat* (*nlength nell*)) = *b*
by (*metis add-cancel-left-right enat-le-plus-same(2) gen-nlength-def is-NNil-ndropn ndropn-NNil ndropn-nnth ndropn-nnth-a nfinite-nlength-enat nlast-NNil nlength-NNil nlength-code plus-enat-simps(1) the-enat.simps zero-enat-def*)
have *2*: *nfinite nell* ⟹ *ndropn n nell* = *NNil b* ⟹ *nlength nell* ≤ *enat n*
by (*metis is-NNil-ndropn nellist.disc(1)*)
have *3*: *nfinite nell* ⟹ *nlength nell* ≤ *enat n* ⟹ *b* = *nnth nell* (*the-enat* (*nlength nell*)) ⟹
ndropn n nell = *NNil* (*nnth nell* (*the-enat* (*nlength nell*)))
by (*metis add.right-neutral is-NNil-ndropn ndropn-nnth-a nellist.collapse(1) nfinite-NNil nlength-NNil nnth-nlast the-enat-0*)
have *4*: ¬*nfinite nell* ⟹
ndropn n nell = (*NNil b*) ⟷
nnth nell (*the-enat*(*nlength nell*)) = *b* ∧ *nlength nell* ≤ (*enat n*)
by (*metis enat-ile is-NNil-ndropn nellist.disc(1) nfinite-conv-nlength-enat*)
show *?thesis*
using *1 2 3 4* **by** *fastforce*
qed

lemma *ntl-ndropn*:
ntl(*ndropn n nell*) = *ndropn n* (*ntl nell*)
by (*simp add*: *funpow-swap1 ndropn-ntl*)

lemma *nfinite-ndropn-a*:
assumes *nfinite nell*
shows *nfinite*(*ndropn n nell*)
using *assms*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case* **by** *auto*
next

case (*Suc n*)
then show *?case* **by** (*simp add*: *ndropn-ntl*)
qed

lemma *nfinite-ndropn-b*:
assumes *nfinite*(*ndropn n nell*)
shows *nfinite nell*
using *assms*
proof (*induct ys*≡*ndropn n nell arbitrary*: *n nell rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*metis enat-ile ndropn-eq-NNil nfinite-conv-nlength-enat*)
next
case (*NCons x nell*)
then show *?case* **by** (*metis nellist.sel*(*5*) *nfinite-ntl ntl-ndropn*)
qed

lemma *nfinite-ndropn*[*simp*]:
nfinite(*ndropn n nell*) = *nfinite nell*
using *nfinite-ndropn-a nfinite-ndropn-b* **by** *blast*

lemma *ndropn-ndropn*:
ndropn m (*ndropn n nell*) = *ndropn* (*n*+*m*) *nell*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
by (*metis add-Suc ndropn-NNil ndropn-Suc nellist.case-eq-if*)
qed

lemma *ndropn-nlast*:
nfinite nell ⟹ *ndropn* (*the-enat*(*nlength nell*)) *nell* = (*NNil* (*nlast nell*))
by (*metis add.left-neutral enat.simps*(*3*) *enat-the-enat ndropn-eq-NNil ndropn-nnth ndropn-nnth-a*
nfinite-conv-nlength-enat order-refl)

lemma *ndropn-nfirst*:
nfirst (*ndropn n nell*) = (*nnth nell n*)
by *transfer*
(*metis lhd-ldropn llist-of-nellist-ndropn lnull-ldropn not-le-imp-less*
not-lnull-conv-llist-of-nellist)

lemma *ndropn-all*:
nlength nell ≤ *enat n* ⟹ *ndropn n nell* = (*NNil* (*nlast nell*))
by (*metis enat-ile ndropn-eq-NNil ndropn-nlast nlength-eq-enat-nfiniteD*)

lemma *ndropn-nappend1*:
nfinite nellx ⟹ *n* ≤ *nlength nelly* ⟹
ndropn (*Suc*(*the-enat* (*nlength nellx*) + *n*)) (*nappend nellx nelly*) = *ndropn n nelly*
proof (*induct arbitrary*: *nelly n rule*: *nfinite-induct*)

case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons x nell*)
then show *?case*
by (*metis ab-semigroup-add-class.add-ac(1) eSuc-enat nappend-NCons ndropn-Suc-NCons nfinite-nlength-enat nlength-NCons plus-1-eq-Suc the-enat.simps*)
qed

lemma *ndropn-nappend2*:
enat n ≤ (*nlength nellx*) ⟹ *ndropn n* (*nappend nellx nelly*) = *nappend* (*ndropn n nellx*) *nelly*
proof (*induct n arbitrary*: *nellx nelly*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis*
using *Suc.prems enat-0-iff*(*1*) **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis*
by (*metis Suc.hyps Suc.prems eSuc-enat eSuc-ile-mono nappend-NCons ndropn-Suc-NCons nlength-NCons*)
qed
qed

lemma *ndropn-nappend3*:
nlength nellx < *enat n* ⟹
ndropn n (*nappend nellx nelly*) = *ndropn* (*n* − (*the-enat* (*eSuc*(*nlength nellx*)))) *nelly*
proof (*induct n arbitrary*: *nellx nelly*)
case *0*
then show *?case* **using** *zero-enat-def* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis*
by (*metis add-diff-cancel-left′ nappend-NNil ndropn-Suc-NCons nlength-NNil one-eSuc one-enat-def plus-1-eq-Suc the-enat.simps*)
next
case (*NCons x21 x22*)
then show *?thesis*
by (*metis Extended-Nat.eSuc-mono Suc.hyps Suc.prems add-diff-cancel-left′ diff-Suc-eq-diff-pred eSuc-enat enat-ord-code*(*4*) *nappend-NCons ndropn-Suc-NCons nlength-NCons order-less-imp-not-less plus-1-eq-Suc the-enat-eSuc*)
qed

qed

lemma *nset-ndropn*:
nset (*ndropn n nell*) ⊆ *nset nell*
by *transfer* (*simp add*: *lset-ldropn-subset*)

lemma *ndropn-nmap*:
ndropn n (*nmap f nell*) = *nmap f* (*ndropn n nell*)
by *transfer*
(*auto*,
metis eSuc-epred enat-the-enat iless-Suc-eq infinity-ileE leD llength-eq-0 min.cobounded1
min.cobounded2)

lemma *nappend-ntaken-ndropn*:
assumes (*Suc k*) ≤ *nlength nell*
shows *nappend* (*ntaken k nell*) (*ndropn* (*Suc k*) *nell*) = *nell*
using *assms*
by *transfer* (*simp add*: *min-absorb1*)

lemma *nfirst-eq-nnth-zero*:
nfirst nell = *nnth nell 0*
by (*metis ndropn-0 ndropn-nfirst*)

1.2.17 *nzip*

lemma *nzip-nhd*:
¬*is-NNil nellx* ∧ ¬*is-NNil nelly* ⟹ *nhd* (*nzip nellx nelly*) = (*nhd nellx*, *nhd nelly*)
by *transfer*
(*auto simp add*: *lzip-eq-LCons-conv*,
metis lzip-eq-LNil-conv)

lemma *nzip-ntl*:
¬*is-NNil nellx* ∧ ¬*is-NNil nelly* ⟹ *ntl*(*nzip nellx nelly*) = *nzip* (*ntl nellx*) (*ntl nelly*)
by *transfer*
(*auto*,
metis lhd-LCons-ltl ltl-lzip ltl-simps(*2*) *lzip-eq-LNil-conv*,
metis (*full-types*) *lhd-LCons-ltl lnull-def*,
metis lhd-LCons-ltl llist.collapse(*1*))

lemma *nzip-simps* [*simp*, *code*, *nitpick-simp*]:
nzip (*NNil b*) *nelly* = (*NNil* (*b*, (*nnth nelly 0*)))
nzip nellx (*NNil b*) = (*NNil* ((*nnth nellx 0*), *b*))
nzip (*NCons x nellx*) (*NCons y nelly*) = *NCons* (*x*, *y*) (*nzip nellx nelly*)
apply *transfer*
apply (*auto simp add*: *not-lnull-conv*)
apply (*metis lnth-0 min-enat-simps*(*3*) *the-enat-0 zero-enat-def*)
apply *transfer*
apply (*auto simp add*: *not-lnull-conv*)
apply (*metis lnth-0 min-enat-simps*(*3*) *the-enat-0 zero-enat-def*)

apply *transfer*
by (*auto simp add*: *not-lnull-conv*)

lemma *is-NNil-nzip* [*simp*]:
is-NNil (*nzip nellx nelly*) ⟷ (*is-NNil nellx*) ∨ (*is-NNil nelly*)
by *transfer*
(*auto simp add*: *lzip-eq-LCons-conv not-lnull-conv*,
metis lzip-eq-LNil-conv)

lemma *nzip-eq-NNil-conv*:
nzip nellx nelly = (*NNil* (*x*, *y*)) ⟷
((*is-NNil nellx*) ∨ (*is-NNil nelly*)) ∧ (*nnth nellx 0*) = *x* ∧ (*nnth nelly 0*) = *y*
by *auto*
(*metis is-NNil-nzip nellist.disc*(*1*),
metis Pair-inject is-NNil-nzip nellist.collapse(*1*) *nellist.disc*(*1*) *nlast-NNil nzip-simps*(*1*)
nzip-simps(*2*),
metis Pair-inject is-NNil-def is-NNil-nzip nellist.inject(*1*) *nzip-simps*(*1*) *nzip-simps*(*2*),
metis nellist.collapse(*1*) *nnth-NNil nzip-simps*(*1*),
metis nellist.collapse(*1*) *nnth-NNil nzip-simps*(*2*))

lemma *nzip-eq-NCons-conv*:
nzip nellx nelly = (*NCons z zs*) ⟷
(∃ *x nellx′ y nelly′*. *nellx* = (*NCons x nellx′*) ∧ *nelly* = (*NCons y nelly′*) ∧
z = (*x*, *y*) ∧ *zs* = (*nzip nellx′ nelly′*))
by (*cases nellx nelly rule*: *nellist.exhaust*[*case-product nellist.exhaust*])
auto

lemma *nzip-nappend*:
nlength nellx = *nlength nellu*
⟹ *nzip* (*nappend nellx nelly*) (*nappend nellu nellv*) =
nappend (*nzip nellx nellu*) (*nzip nelly nellv*)
by *transfer*
(*simp*,
meson co.enat.expand llength-eq-0 lzip-lappend)

lemma *nlength-nzip* [*simp*]:
nlength (*nzip nellx nelly*) = (*min* (*nlength nellx*) (*nlength nelly*))
by *transfer simp*

lemma *ntake-nzip*:
ntake n (*nzip nellx nelly*) = *nzip* (*ntake n nellx*) (*ntake n nelly*)
by *transfer*
(*simp add*: *ltake-lzip*)

lemma *ntaken-nzip*:
ntaken n (*nzip nellx nelly*) = *nzip* (*ntaken n nellx*) (*ntaken n nelly*)
by *transfer*
(*simp add*: *enat-0-iff*(*1*) *ltake-lzip*)

lemma *ndropn-nzip* [*simp*]:

$n \leq nlength\ nellx \wedge n \leq nlength\ nelly \implies$
$ndropn\ n\ (nzip\ nellx\ nelly) = nzip\ (ndropn\ n\ nellx)\ (ndropn\ n\ nelly)$
by *transfer*
(*auto simp add*: *min-def*,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0)

lemma *nzip-niterates*:
$nzip\ (niterates\ f\ x)\ (niterates\ g\ y) = niterates\ (\lambda(x,y).\ (f\ x,\ g\ y))\ (x,\ y)$
by *transfer*
(*simp add*: *lzip-iterates*)

lemma *nnth-nzip*:
assumes $n \leq nlength\ nellx$
$n \leq nlength\ nelly$
shows $nnth\ (nzip\ nellx\ nelly)\ n = (nnth\ nellx\ n,\ nnth\ nelly\ n)$
using *assms*
by *transfer*
(*auto simp add*: *min-def*,
metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lzip)

lemma *nset-nzip*:
$nset\ (nzip\ nellx\ nelly) =$
$\{\ (nnth\ nellx\ n,\ nnth\ nelly\ n) \mid n.\ n \leq min\ (nlength\ nellx)\ (nlength\ nelly)\}$
by *transfer*
(*auto simp add*: *in-lset-conv-lnth*,
metis Pair-inject co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lzip min.orderE
the-enat.simps,
metis eSuc-epred iless-Suc-eq llength-eq-0 lnth-lzip)

lemma *nset-nzipD1*:
$(x,\ y) \in nset\ (nzip\ nellx\ nelly) \implies x \in nset\ nellx$
by *transfer*
(*meson lset-lzipD1*)

lemma *nset-nzipD2*:
$(x,\ y) \in nset\ (nzip\ nellx\ nelly) \implies y \in nset\ nelly$
by *transfer*
(*meson lset-lzipD2*)

lemma *nset-nzip-same* [*simp*]:
$nset\ (nzip\ nellx\ nellx) = (\lambda\ x.\ (x,\ x))\ `\ nset\ nellx$
by *transfer*
simp

lemma *nfinite-nzip* [*simp*]:
$nfinite\ (nzip\ nellx\ nelly) \longleftrightarrow nfinite\ nellx \vee nfinite\ nelly$
by *transfer*
simp

lemma *nzip-eq-nappend-conv*:
assumes *eq*: *nzip nellx nelly = nappend nellu nellv*
shows $\exists$ *nellx' nellx'' nelly' nelly''*.
nellx = nappend nellx' nellx'' ∧ nelly = nappend nelly' nelly'' ∧
nlength nellx' = nlength nelly' ∧
nellu = nzip nellx' nelly' ∧ nellv = nzip nellx'' nelly''
using *assms*
apply *transfer*
using *lzip-eq-lappend-conv*
by (*auto simp add*: *lzip-eq-lappend-conv*)
fastforce

lemma *nzip-nmap* [*simp*]:
nzip (*nmap f nellx*) (*nmap g nelly*) = *nmap* (λ(*x*, *y*). (*f x*, *g y*)) (*nzip nellx nelly*)
by *transfer auto*

lemma *nzip-nmap1*:
nzip (*nmap f nellx*) *nelly* = *nmap* (λ(*x*, *y*). (*f x*, *y*)) (*nzip nellx nelly*)
by *transfer*
(*simp add*: *lzip-lmap1*)

lemma *nzip-nmap2*:
nzip nellx (*nmap f nelly*) = *nmap* (λ(*x*, *y*). (*x*, *f y*)) (*nzip nellx nelly*)
by *transfer*
(*simp add*: *lzip-lmap2*)

lemma *nmap-fst-nzip-conv-ntake*:
nmap fst (*nzip nellx nelly*) = *ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nellx*
by *transfer*
(*auto*,
metis co.enat.exhaust-sel llength-eq-0 lmap-fst-lzip-conv-ltake min-eSuc-eSuc)

lemma *nmap-snd-nzip-conv-ntake*:
nmap snd (*nzip nellx nelly*) = *ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nelly*
by *transfer*
(*auto*,
metis co.enat.exhaust-sel llength-eq-0 lmap-snd-lzip-conv-ltake min-eSuc-eSuc)

lemma *nzip-conv-nzip-ntake-min-nlength*:
nzip nellx nelly =
nzip (*ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nellx*)
(*ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nelly*)
by *transfer*
(*auto*,
metis co.enat.exhaust-sel epred-min i0-lb llength-lzip ltake-all ltake-lzip order-refl)

lemma *nellist-all2-conv-nzip*:
nellist-all2 P nellx nelly ⟷
nlength nellx = nlength nelly ∧ (∀ (*x*, *y*) ∈ *nset*(*nzip nellx nelly*). *P x y*)
using *nset-nzip*[*of nellx nelly*]

by (*auto simp add*: *nellist-all2-conv-all-nnth*)
blast

lemma *nellist-all2-all-nnthI*:
assumes *nlength nellx* = *nlength nelly*
⋀*n*. *enat n* ≤ *nlength nellx* ⟹ *P* (*nnth nellx n*) (*nnth nelly n*)
shows *nellist-all2 P nellx nelly*
using *assms* **by** (*simp add*: *nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nsetD1*:
assumes *nellist-all2 P nellx nelly*
x ∈ *nset nellx*
shows ∃ *y* ∈ *nset nelly*. *P x y*
using *assms*
by (*metis in-nset-conv-nnth nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nsetD2*:
assumes *nellist-all2 P nellx nelly*
y ∈ *nset nelly*
shows ∃ *x* ∈ *nset nellx*. *P x y*
using *assms*
by (*metis in-nset-conv-nnth nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nzipI*:
assumes *nellist-all2 P nellx nelly*
nellist-all2 P′ nellx′ nelly′
shows *nellist-all2* (*rel-prod P P′*) (*nzip nellx nellx′*) (*nzip nelly nelly′*)
using *assms*
proof (*coinduction arbitrary*: *nellx nellx′ nelly nelly′*)
case (*ilist-all2 nx nx′ ny ny′*)
then show *?case*
proof −
have *1*: *is-NNil* (*nzip nx nx′*) = *is-NNil* (*nzip ny ny′*)
by (*metis ilist-all2*(*1*) *ilist-all2*(*2*) *is-NNil-nzip nellist-all2-is-NNilD*)
have *2*: (*is-NNil* (*nzip nx nx′*) ⟹
is-NNil (*nzip ny ny′*) ⟹
rel-prod P P′ (*nlast* (*nzip nx nx′*)) (*nlast* (*nzip ny ny′*)))
by (*metis* (*no-types*, *lifting*) *enat-min-eq-0-iff ilist-all2*(*1*) *ilist-all2*(*2*)
is-NNil-nzip min-def-raw nellist.sel(*1*) *nellist-all2-conv-all-nnth*
nzip-eq-NNil-conv order-refl rel-prod-inject zero-enat-def)
have *3*: (¬ *is-NNil* (*nzip nx nx′*) ⟹
¬ *is-NNil* (*nzip ny ny′*) ⟹
(∃ *nellx nellx′ nelly nelly′*.
ntl (*nzip nx nx′*) = *nzip nellx nellx′* ∧
ntl (*nzip ny ny′*) = *nzip nelly nelly′* ∧
nellist-all2 P nellx nelly ∧ *nellist-all2 P′ nellx′ nelly′*) ∨
nellist-all2 (*rel-prod P P′*) (*ntl* (*nzip nx nx′*)) (*ntl* (*nzip ny ny′*))))
by (*auto simp add*: *nzip-eq-NCons-conv dest*: *nellist-all2-nhdD intro*: *nellist-all2-ntlI*)
(*meson ilist-all2*(*1*) *ilist-all2*(*2*) *nellist-all2-ntlI nzip-ntl*)
have *4*: ¬ *is-NNil* (*nzip nx nx′*) ⟶

```
        ¬ is-NNil (nzip ny ny') ⟶
        rel-prod P P' (nhd (nzip nx nx')) (nhd (nzip ny ny'))
    by (simp add: ilist-all2(1) ilist-all2(2) nellist-all2-nhdD nzip-nhd)
  have 5: (¬ is-NNil (nzip nx nx') ⟶
        ¬ is-NNil (nzip ny ny') ⟶
         rel-prod P P' (nhd (nzip nx nx')) (nhd (nzip ny ny')) ∧
         ((∃ nellx nellx' nelly nelly'.
            ntl (nzip nx nx') = nzip nellx nellx' ∧
            ntl (nzip ny ny') = nzip nelly nelly' ∧
            nellist-all2 P nellx nelly ∧ nellist-all2 P' nellx' nelly') ∨
            nellist-all2 (rel-prod P P') (ntl (nzip nx nx')) (ntl (nzip ny ny'))))
    using 3 4 by blast
  show ?thesis
    using 1 2 3 4 by presburger
 qed
qed

lemma ndistinct-nzipI1:
 ndistinct nellx ⟹ ndistinct (nzip nellx nelly)
by transfer
  (simp add: ldistinct-lzipI1)

lemma ndistinct-nzipI2:
 ndistinct nelly ⟹ ndistinct (nzip nellx nelly)
by transfer
  (simp add: ldistinct-lzipI2)
```

1.2.18 *niterates*

```
lemma niterates-not-is-NNil [nitpick-simp, simp]:
 ¬ is-NNil (niterates f x)
by transfer
  (metis lfinite-LConsI lfinite-code(1) lfinite-iterates)

lemma nhd-niterates [code, simp, nitpick-simp]:
 nhd(niterates f x) = x
by transfer
  (metis lfinite-LConsI lfinite-LNil lfinite-iterates lhd-iterates)

lemma ntl-niterates [code, simp, nitpick-simp]:
 ntl(niterates f x) = niterates f (f x)
by transfer
  (simp,
   metis lfinite-LConsI lfinite-LNil lfinite-iterates)

lemma nfinite-niterates [iff]:
 ¬ nfinite (niterates f x)
by transfer  simp

lemma niterates-nmap:
```

```
  niterates f x = NCons x (nmap f (niterates f x))
by transfer
   (meson iterates.disc-iff iterates-lmap)

lemma [simp]:
  fixes f :: 'a ⇒ 'a
  shows is-NNil-funpow-nmap: is-NNil ((nmap f ^^ n) nellx) ⟷ is-NNil nellx
  and nhd-funpow-nmap: ¬ is-NNil nellx ⟹ nhd ((nmap f ^^ n) nellx) = (f ^^ n) (nhd nellx)
  and ntl-funpow-nmap: ¬ is-NNil nellx ⟹ ntl ((nmap f ^^ n) nellx) = (nmap f ^^ n) (ntl nellx)
by (induct n) simp-all

lemma niterates-equality:
 assumes h:  ⋀x. h x = NCons x (nmap f (h x))
 shows    h = niterates f
proof −
 { fix x
   have ¬ is-NNil (h x) nhd (h x) = x  ntl (h x) = nmap f (h x)
    by (subst h, simp)+ }
 note [simp] = this
 { fix x
   define n :: nat where n = 0
   have (nmap f ^^ n) (h x) = (nmap f ^^ n) (niterates f x)
     proof (coinduction arbitrary: n)
     case (Eq-nellist nn)
     then show ?case
       proof −
       have 1: is-NNil ((nmap f ^^ nn) (h x)) = is-NNil ((nmap f ^^ nn) (niterates f x))
         by auto
       have 2: (is-NNil ((nmap f ^^ nn) (h x)) ⟶
                 is-NNil ((nmap f ^^ nn) (niterates f x)) ⟶
                 nlast ((nmap f ^^ nn) (h x)) = nlast ((nmap f ^^ nn) (niterates f x)))
         by simp
       have 3: (¬ is-NNil ((nmap f ^^ nn) (h x)) ⟶
                 ¬ is-NNil ((nmap f ^^ nn) (niterates f x)) ⟶
                 nhd ((nmap f ^^ nn) (h x)) = nhd ((nmap f ^^ nn) (niterates f x)))
         by simp
       have 4: (¬ is-NNil ((nmap f ^^ nn) (h x)) ⟶
                 ¬ is-NNil ((nmap f ^^ nn) (niterates f x)) ⟶
                 ( ntl ((nmap f ^^ nn) (h x)) = (nmap f ^^ (Suc nn)) (h x) ∧
                     ntl ((nmap f ^^ nn) (niterates f x)) = (nmap f ^^ (Suc nn)) (niterates f x)))
         by (metis ‹⋀x. ¬ is-NNil (h x)› ‹⋀x. ntl (h x) = nmap f (h x)› funpow-simps-right(2)
             nellist.sel(5) niterates-nmap niterates-not-is-NNil ntl-funpow-nmap o-apply)
       show ?thesis using 1 2 3 4 by blast
       qed
     qed
   }
   thus ?thesis by auto
qed

lemma nlength-niterates [simp]:
```

nlength (*niterates f x*) = ∞
by *transfer auto*

lemma *ndropn-niterates*:
ndropn n (*niterates f x*) = *niterates f* ((*f* ^^ *n*) *x*)
by *transfer*
(*simp add*: *ldropn-iterates*)

lemma *nnth-niterates* [*simp*]:
nnth (*niterates f x*) *n* = (*f* ^^ *n*) *x*
by *transfer auto*

lemma *nset-niterates*:
nset (*niterates f x*) = { (*f* ^^ *n*) *x*|*n*. *True*}
by *transfer*
(*metis lset-iterates*)

lemma *nnth-niterates-Suc*:
nnth (*niterates Suc 0*) *i* = *i*
proof (*induct i*)
case *0*
then show *?case*
by *force*
next
case (*Suc i*)
then show *?case* **by** *simp*
qed

1.2.19 Filtering non-empty lazy lists *nfilter*

lemma *nfilter-NNil* [*simp*]:
shows *nfilter P* (*NNil b*) = *NNil b*
by *transfer auto*

lemma *nfilter-True* [*simp*]:
shows *nfilter* (λ*x*. *True*) *nell* = *nell*
by *transfer auto*

lemma *nfilter-False-finite*:
assumes *nfinite nell*
shows *nfilter* (λ *x*. *False*) *nell* = *nell*
using *assms* **by** *transfer auto*

lemma *nfilter-NCons* [*simp*]:
assumes (∃ *x* ∈ *nset nell*. *P x*)
shows *nfilter P* (*NCons x nell*) = (*if P x then NCons x* (*nfilter P nell*) *else nfilter P nell*)
using *assms* **by** *transfer auto*

lemma *nfilter-NCons-a* [*simp*]:
assumes ¬(∃ *x* ∈ *nset nell*. *P x*)

```
      P x
 shows nfilter P (NCons x nell) = (NNil x)
using assms by transfer auto

lemma nfilter-expand:
assumes ∃ x ∈ nset nell. P x
shows  nfilter P nell =
       (if is-NNil nell then nell
        else
        (if (∃x ∈ nset(ntl nell). P x) then
          (if P (nhd nell) then (NCons (nhd nell) (nfilter P (ntl nell)))
                           else (nfilter P (ntl nell)) )
         else (NNil (nhd nell)) ))
using assms by (cases nell) auto

lemma nset-nfilter:
 assumes ∃ x ∈ nset nell. P x
 shows nset (nfilter P nell) = nset nell ∩ {xa. P xa}
using assms
by transfer auto

lemma exist-conj:
assumes ∃ x ∈ nset (nfilter Q nell). P x
       ∃ x ∈ nset nell. Q x
shows ∃ x ∈ nset nell. P x ∧ Q x
 using assms
by transfer (auto split: if-split-asm)

lemma nfilter-nfilter [simp]:
assumes ∃ x ∈ nset (nfilter Q nell). P x
       ∃ x ∈ nset nell. Q x
shows nfilter P (nfilter Q nell) = nfilter (λx. P x ∧ Q x) nell
using assms
proof (transfer fixing: P Q)
fix xsa :: 'a llist
assume ¬ lnull xsa ∧ xsa = xsa
assume a1: Bex (lset (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa)) P
assume Bex (lset xsa) Q
then have ¬ lnull (lfilter Q xsa)
by simp
then have lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa) = lfilter P (lfilter Q xsa) ∧
          ¬ lnull (lfilter P (lfilter Q xsa)) ∧
          ¬ lnull (if lnull (lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa))
                   then if lnull (lfilter Q xsa) then xsa else lfilter Q xsa
                   else lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa))
using a1 by (auto split: if-split-asm)
then show ¬ lnull (if lnull (lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa))
                   then if lnull (lfilter Q xsa) then xsa else lfilter Q xsa
                    else lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa)) ∧
           (if lnull (lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa))
```

then if lnull (*lfilter Q xsa*) *then xsa else lfilter Q xsa*
else lfilter P (*if lnull* (*lfilter Q xsa*) *then xsa else lfilter Q xsa*)) =
(*if lnull* (*lfilter* (λ*a*. *P a* ∧ *Q a*) *xsa*) *then xsa else lfilter* (λ*a*. *P a* ∧ *Q a*) *xsa*)
using *lfilter-lfilter* **by** *auto*
qed

lemma *length-nfilter-le* [*simp*]:
nlength (*nfilter P nell*) ≤ *nlength nell*
by *transfer* (*simp add*: *epred-le-epredI llength-lfilter-ile*)

lemma *nfilter-nnth*:
assumes (∃ *x* ∈ *nset nell*. *P x*)
i≤ *nlength* (*nfilter P nell*)
shows (∃ *k* ≤ *nlength nell*. *nnth*(*nfilter P nell*) *i* = *nnth nell k*)
using *assms nset-nfilter*[*of nell P*]
using *in-nset-conv-nnth*
by (*metis Int-iff*)

lemma *nfilter-nappend1*:
assumes ∀ *x*∈*nset nell*. ¬ *P x*
nfinite nell
∃ *y* ∈ *nset nell1*. *P y*
shows *nfilter P* (*nappend nell nell1*) = *nfilter P nell1*
using *assms* **by** *transfer auto*

lemma *nfilter-nappend2*:
assumes ∀ *x*∈*nset nell*. ¬ *P x*
∃ *y*∈*nset nell1*. *P y*
shows *nfilter P* (*nappend nell1 nell*) = *nfilter P nell1*
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
meson in-lset-lappend-iff,
metis lappend-LNil2 lappend-inf lfilter-empty-conv lfilter-lappend-lfinite)

lemma *nfilter-nappend* [*simp*]:
assumes (∃ *x* ∈ *nset* (*nappend nell nell1*). *P x*)
(∃ *x* ∈ *nset nell*. *P x*)
(∃ *x* ∈ *nset nell1*. *P x*)
shows *nfilter P* (*nappend nell nell1*) =
(*if nfinite nell then nappend* (*nfilter P nell*) (*nfilter P nell1*)
else (*nfilter P nell*))
proof (*cases nfinite nell*)
case *True*
then show *?thesis* **using** *assms* **by** *transfer auto*
next
case *False*
then show *?thesis* **using** *assms* **by** *transfer* (*auto simp add*: *lappend-inf*)
qed

lemma *nfilter-nmap*:
shows *nfilter P (nmap f nell) = nmap f (nfilter (P∘f) nell)*
by *transfer (auto simp add: lfilter-lmap)*

lemma *nlength-nfilter-nmap[simp]*:
shows *nlength (nfilter P (nmap f nell)) = nlength(nfilter (P ∘ f) nell)*
by *(simp add: nfilter-nmap)*

lemma *nfilter-is-subset [simp]*:
assumes *∃ x ∈ nset nell. P x*
shows *nset (nfilter P nell) ≤ nset nell*
using *assms* **by** *(simp add: nset-nfilter)*

lemma *nfilter-cong[fundef-cong]*:
assumes *nell = nell1*
(⋀x. x ∈ nset nell1 ⟹ P x = Q x)
shows *nfilter P nell = nfilter Q nell1*
using *assms* **by** *transfer auto*

lemma *nset-nappend*:
nset (nappend nell nell1) = (if nfinite nell then nset nell ∪ nset nell1 else nset nell)
by *transfer (simp add: lappend-inf)*

lemma *split-nellist-nappend*:
assumes *∃ i ≤ nlength nell. nnth nell i = x ∧ P (nnth nell i) ∧*
(∀ j. j ≠ i ∧ j ≤ nlength nell ⟶ ¬ P(nnth nell j))
shows *P x ∧*
(nell = (NNil x) ∨
(∃ vs. nell = (NCons x vs) ∧ (∀ v ∈ nset vs. ¬ P v)) ∨
(∃ us. nell = nappend us (NNil x) ∧ nfinite us ∧ (∀ u ∈ nset us. ¬ P u)) ∨
(∃ us vs . nell = nappend us (NCons x vs) ∧ nfinite us ∧
(∀ u ∈ nset us. ¬ P u) ∧ (∀ v ∈ nset vs. ¬ P v))
)
proof *−*
obtain *i* **where** *1: i ≤ nlength nell ∧ nnth nell i = x ∧ P (nnth nell i) ∧*
(∀ j. j ≠ i ∧ j ≤ nlength nell ⟶ ¬ P(nnth nell j))
using *assms* **by** *auto*
have *01: (∀ j. (j<i ∨ i<j) ∧ j ≤ nlength nell ⟶ ¬ P(nnth nell j))*
using *1*
by *auto*
have *2: i=0 ∧ nlength nell = 0 ⟹ P x ∧ nell = (NNil x)*
by *(metis 1 ndropn-0 ndropn-eq-NNil the-enat-0 zero-enat-def)*
have *3: i=0 ∧nlength nell > 0 ⟹ P x ∧ nell = (NCons x (ntl nell)) ∧ (∀ v ∈ nset (ntl nell). ¬ P v)*
using *1 in-nset-conv-nnth[of - ntl nell]*
by *(metis Suc-ile-eq iless-Suc-eq nat.simps(3) nellist.disc(2) nellist.exhaust-sel nlength-NCons*
nlength-NNil nnth-0-conv-nhd nnth-ntl not-less-iff-gr-or-eq)
have *4: i=0 ∧nlength nell > 0 ⟹ P x ∧ (∃ vs. nell = (NCons x vs) ∧ (∀ v ∈ nset vs. ¬ P v))*
using *3* **by** *auto*
have *5: i >0 ∧ i = nlength nell ∧ nfinite nell ⟹ P x ∧ nell = nappend (ntaken (i−1) nell) (NNil x)*

```
  using 1
  by transfer
    (auto,
     metis eSuc-epred lappend-lbutlast-llast-id-lfinite lbutlast-conv-ltake llast-conv-lnth
     llength-eq-0 the-enat.simps)
 have 6: i >0 ∧ i = nlength nell ∧ nfinite nell ⟹ (∀ u ∈ nset (ntaken (i−1) nell). ¬ P u)
  using 1
  by transfer
     (auto,
      metis in-lset-conv-lnth lbutlast-conv-ltake less-imp-le llength-lbutlast lnth-ltake
      min.strict-order-iff)
 have 7: i >0 ∧ i = nlength nell ∧ nfinite nell ⟹ P x ∧ (∃ us. nell = nappend us (NNil x) ∧
         nfinite us ∧ (∀ u ∈ nset us. ¬ P u))
  using 5 6 by auto
 have 8: i >0 ∧ i < nlength nell ⟹
          P x ∧ nell = nappend (ntaken (i−1) nell) (NCons x (ndropn (i+1) nell))
  using 1
  by transfer
    (auto,
     metis co.enat.collapse eSuc-enat ileI1 iless-Suc-eq lappend-ltake-enat-ldropn
     ldropn-Suc-conv-ldropn llength-eq-0 min.absorb1 the-enat.simps)
 have 9: i >0 ∧ i < nlength nell ⟹ nfinite (ntaken (i−1) nell)
  using enat-ord-code(4) nfinite-ntaken by blast
 have 10: i >0 ∧ i < nlength nell ⟹ (∀ u ∈ nset (ntaken (i−1) nell). ¬ P u)
  using 01 in-nset-conv-nnth[of - (ntaken (i−1) nell) ] by simp
  (metis Suc-pred le-imp-less-Suc min.orderE ntaken-nnth)
 have 11: i >0 ∧ i < nlength nell ⟹ (∀ v ∈ nset (ndropn (i+1) nell). ¬ P v)
  using 1 01 in-nset-conv-nnth[of - (ndropn (i+1) nell) ]
  by simp
    (metis Suc-ile-eq iless-Suc-eq is-NNil-ndropn le-add1 le-imp-less-Suc ndropn-Suc-conv-ndropn
     ndropn-ndropn ndropn-nlength nlength-NCons not-less)
 have 12: i >0 ∧ i < nlength nell ⟹ (∃ us vs . nell = nappend us (NCons x vs) ∧ nfinite us ∧
          (∀ u ∈ nset us. ¬ P u) ∧ (∀ v ∈ nset vs. ¬ P v))
  using 8 9 10 11 by blast
 show ?thesis
  by (metis 1 12 2 4 7 dual-order.order-iff-strict neq0-conv nlength-eq-enat-nfiniteD
      zero-enat-def)
qed

lemma nellist-split-2-first:
 assumes 0 < nlength nell
shows nell = (NCons (nnth nell 0) (ntl nell))
using assms by (metis ndropn-0 ndropn-Suc-conv-ndropn nellist.sel(5) zero-enat-def)

lemma nellist-split-2-last:
 assumes 0< i
         i = nlength nell
          nfinite nell
 shows nell = nappend (ntaken (i−1) nell) (NNil (nnth nell i))
using assms
```

by *transfer*
(*simp*,
metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat llength-eq-0 ltake-Suc-conv-snoc-lnth ltake-all order-refl the-enat.simps)

lemma *nellist-split-3*:
assumes $0 < i$
$i <$ *nlength nell*
shows *nell* = *nappend* (*ntaken* (*i*−*1*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*i*+*1*) *nell*))
using *assms* **by** *transfer*
(*auto*,
metis eSuc-enat eSuc-epred ileI1 iless-Suc-eq lappend-ltake-enat-ldropn ldropn-Suc-conv-ldropn less-imp-le llength-eq-0 min-def the-enat.simps)

lemma *NNil-eq-nfilterD*:
assumes $\exists\ x \in$ *nset nell*. *P x*
(*NNil x*) = *nfilter P nell*
shows ($\exists$ *us vs* . (*nell* = (*NNil x*) $\vee$
(*nell* = (*NCons x vs*) $\wedge$ ($\forall\ v \in$ *nset vs*. $\neg$ *P v*)) $\vee$
(*nell* = *nappend us* (*NNil x*) $\wedge$ *nfinite us* $\wedge$ ($\forall\ u \in$ *nset us*. $\neg$ *P u*)) $\vee$
(*nell* = *nappend us* (*NCons x vs*) $\wedge$ *nfinite us* $\wedge$
($\forall\ u \in$ *nset us*. $\neg$ *P u*) $\wedge$ ($\forall\ v \in$ *nset vs*. $\neg$ *P v*))
) $\wedge$ *P x*)
proof −
have *1*: ($\exists\ i \leq$ *nlength nell*. *P* (*nnth nell i*))
by (*metis assms*(*1*) *in-nset-conv-nnth*)
obtain *i* **where** *2*: $i \leq$ *nlength nell* $\wedge$ *P* (*nnth nell i*)
using *1* **by** *auto*
have *3*: *i* = *0* $\wedge$ *nlength nell* = *0* $\Longrightarrow$ *P x* $\wedge$ *nell* = (*NNil x*)
by (*metis 2 assms*(*2*) *ndropn-0 ndropn-eq-NNil nfilter-NNil the-enat-0 zero-enat-def*)
have *4*: *i*=*0* $\wedge$*nlength nell* > *0* $\Longrightarrow$*P x* $\wedge$ *nell* = (*NCons x* (*ntl nell*)) $\wedge$ ($\forall\ v \in$ *nset* (*ntl nell*). $\neg$ *P v*)
using *2 assms nellist-split-2-first*[*of nell*] *nfilter-expand*[*of nell P*]
by (*metis nellist.distinct*(*1*) *nellist.sel*(*3*) *nlast-NNil*)
have *5*: *i*=*0* $\wedge$*nlength nell* > *0* $\Longrightarrow$ *P x* $\wedge$ ($\exists$ *vs*. *nell* = (*NCons x vs*) $\wedge$ ($\forall\ v \in$ *nset vs*. $\neg$ *P v*))
using *4* **by** *auto*
have *60*: *0*< *i* $\wedge$ *i* = *nlength nell* $\wedge$ *nfinite nell* $\Longrightarrow$ *x*= (*nnth nell i*)
using *2 assms nellist-split-2-last*[*of i nell*]
proof *simp*
assume *a1*: *nell* = *nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NNil* (*nnth nell i*))
assume *a2*: *P* (*nnth nell i*)
assume *a3*: *NNil x* = *nfilter P nell*
assume *a4*: $\exists x \in$*nset nell*. *P x*
have *f5*: $\forall$ *as p asa*. (($\exists$ *a*. (*a*::′*a*) $\in$ *nset as* $\wedge$ *p a*) $\vee$ $\neg$ *nfinite as* $\vee$ ($\forall$ *a*. *a* $\notin$ *nset asa* $\vee$ $\neg$ *p a*))
$\vee$ *nfilter p* (*nappend as asa*) = *nfilter p asa*
by (*metis* (*full-types*) *nfilter-nappend1*)
have *f7*: $\exists$ *a*. *a* $\in$ *nset* (*NNil* (*nnth nell i*)) $\wedge$ *P a*
using *a2* **by** *auto*
have *f8*: *nfilter P* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NNil* (*nnth nell i*))) $\neq$
nappend (*nfilter P* (*ntaken* (*i* − *Suc 0*) *nell*)) (*nfilter P* (*NNil* (*nnth nell i*)))
using *a3 a1* **by** (*metis* (*full-types*) *is-NNil-nappend nellist.disc*(*1*))

have *f9*: *nfinite* (*ntaken* (*i* − *Suc 0*) *nell*)
by *simp*
obtain *aaa* :: *'a* **where**
f9: *aaa* ∈ *nset nell* ∧ *P aaa*
using *a4* **by** *blast*
then have *aaa* ∈ *nset* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NNil* (*nnth nell i*)))
using *a1* **by** *presburger*
then have *f10*: ∀ *a*. *a* ∉ *nset* (*ntaken* (*i* − *Suc 0*) *nell*) ∨ ¬ *P a*
using *f9 f8 f7* **by** (*meson nfilter-nappend nfinite-ntaken*)
then show *?thesis*
using *f9 f7 f5 a3 a1*
proof −
have *NNil* (*nnth nell i*) = *NNil x*
using *nfilter-NNil*[*of P* (*nnth nell i*)] **by** (*metis* (*no-types*) *f10 a1 a3 f5 f7 nfinite-ntaken*)
then show *?thesis*
by *blast*
qed
qed
have *6*: *0*< *i* ∧ *i* = *nlength nell* ∧ *nfinite nell* ⟹ *P x* ∧ *nell* = *nappend* (*ntaken* (*i*−*1*) *nell*) (*NNil x*)
using *2 60 assms nellist-split-2-last*[*of i nell*]
by *blast*
have *7*: *i* >*0* ∧ *i* = *nlength nell* ∧ *nfinite nell* ⟹ (∀ *u* ∈ *nset* (*ntaken* (*i*−*1*) *nell*). ¬ *P u*)
using *assms 6 nfilter-nappend*[*of* (*ntaken* (*i*−*1*) *nell*) - *P*]
by (*metis is-NNil-nappend nellist.disc*(*1*) *nfinite-ntaken split-nellist-a*)
have *8*: *i* >*0* ∧ *i* = *nlength nell* ∧ *nfinite nell* ⟹ *P x* ∧ (∃ *us*. *nell* = *nappend us* (*NNil x*) ∧
nfinite us ∧ (∀ *u* ∈ *nset us*. ¬ *P u*))
using *6 7* **by** *auto*
have *9*: *i* >*0* ∧ *i* < *nlength nell* ⟹
P x ∧ *nell* = *nappend* (*ntaken* (*i*−*1*) *nell*) (*NCons x* (*ndropn* (*i*+*1*) *nell*))
using *2 assms nellist-split-3*[*of i nell*]
nfilter-nappend1[*of* (*ntaken* (*i*−*1*) *nell*) *P* (*NCons x* (*ndropn* (*i*+*1*) *nell*))]
nfilter-nappend[*of* (*ntaken* (*i* − *1*) *nell*) - *P*]
proof *simp*
assume *a1*: *enat i* ≤ *nlength nell* ∧ *P* (*nnth nell i*)
assume *a2*: *NNil x* = *nfilter P nell*
assume *a3*: ∃ *x*∈*nset nell*. *P x*
assume *a4*: *nell* = *nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))
have *f5*: *P* (*nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))) *i*)
using *a1 a4* **by** *auto*
have *f6*: *NNil x* = *nfilter P* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
using *a2 a4* **by** *auto*
have *f7*: ∀ *as p asa*. ((∃ *a*. (*a*::*'a*) ∈ *nset as* ∧ *p a*) ∨ ¬ *nfinite as* ∨ (∀ *a*. *a* ∉ *nset asa* ∨ ¬ *p a*)) ∨
nfilter p (*nappend as asa*) = *nfilter p asa*
by (*metis* (*full-types*) *nfilter-nappend1*)
have *nfilter P* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))) =
nappend (*nfilter P* (*ntaken* (*i* − *Suc 0*) *nell*)) (*nfilter P* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
⟶
nappend (*nfilter P* (*ntaken* (*i* − *Suc 0*) *nell*)) (*nfilter P* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*)

nell))) =
NNil x
using *f6* **by** *presburger*
then have *nfilter P* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
≠
nappend (*nfilter P* (*ntaken* (*i* − *Suc 0*) *nell*)) (*nfilter P* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*)
nell)))
by (*metis* (*no-types*) *is-NNil-nappend nellist.disc*(*1*))
then have *f9*: (∀ *a*. *a* ∉ *nset* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)) ∨ ¬ *P a*) ∨
nfilter P (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
=
nfilter P (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)) ∨
(∀ *a*. *a* ∉ *nset* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*)
nell)))
∨ ¬ *P a*)
using *nfilter-nappend*[*of* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)) *P*]
nfinite-ntaken[*of* (*i* − *Suc 0*) *nell*]
by (*metis f7*)
obtain *aaa* :: *'a* **where** *f10*: *aaa* ∈ *nset nell* ∧ *P aaa*
using *a3* **by** *blast*
then have *f11*: *aaa* ∈ *nset* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*)
nell)))
using *a4* **by** *presburger*
have *f12*: *nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))) *i* ∈
nset (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))
using *a4* **by** *fastforce*
then have *NNil x* = *nfilter P* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))
using *f11 f10 f9 f6 f5* **by** (*metis* (*no-types*))
then have *NNil x* = *nfilter P* (*NCons* (*nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*)
(*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
i) (*ndropn* (*Suc i*) *nell*))
using *a4* **by** *presburger*
then have *f13*: ∀ *a*. *a* ∉ *nset* (*ndropn* (*Suc i*) *nell*) ∨ ¬ *P a*
using *f5* **by** (*metis* (*full-types*) *nellist.distinct*(*1*) *nfilter-NCons*)
have *nfilter P* (*NCons* (*nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*)
(*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
i) (*ndropn* (*Suc i*) *nell*)) = *NNil x*
using ‹*NNil x* = *nfilter P* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))› *a4* **by** *presburger*
then have *x* = *nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))) *i*
using *f13 f5* **by** *simp*
then have *f14*: *x* = *nnth nell i*
using *a4* **by** *presburger*
then have *nell* = *nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons x* (*ndropn* (*Suc i*) *nell*))
using *a4* **by** *blast*
then show *P x* ∧ *nell* = *nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons x* (*ndropn* (*Suc i*) *nell*))
using *f14 a1* **by** *blast*
qed
have *10*: *i* >*0* ∧ *i* < *nlength nell* ⟹ (∀ *u* ∈ *nset* (*ntaken* (*i*−*1*) *nell*). ¬ *P u*)
using *9 assms nfilter-nappend*[*of* (*ntaken* (*i*−*1*) *nell*) - *P*]
by (*metis is-NNil-nappend nellist.disc*(*1*) *nellist.set-intros*(*2*) *nfinite-ntaken*)

have *11*: $i > 0 \wedge i < nlength\ nell \Longrightarrow (\forall\ v \in nset\ (ndropn\ (i+1)\ nell).\ \neg\ P\ v)$
using *9 10 assms nfilter-nappend*[*of* (*ntaken* (*i*−*1*) *nell*) - *P*]
nfilter-nappend1[*of* (*ntaken* (*i*−*1*) *nell*) *P* (*NCons x* (*ndropn* (*i*+*1*) *nell*))]
by (*metis nellist.distinct*(*1*) *nellist.set-intros*(*2*) *nfilter-NCons nfinite-ntaken*)
have *12*: $i > 0 \wedge i < nlength\ nell \Longrightarrow$
$P\ x \wedge (\exists\ us\ vs\ .\ nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
$(\forall\ u \in nset\ us.\ \neg\ P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
using *9 10 11* **using** *assms*(*1*) **by** *fastforce*
show *?thesis*
by (*metis 12 2 3 5 8 dual-order.order-iff-strict neq0-conv nlength-eq-enat-nfiniteD zero-enat-def*)
qed

lemma *nfilter-eq-NNilD*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$nfilter\ P\ nell\ = (NNil\ x)$
shows $(\exists\ us\ vs\ .\ (nell = (NNil\ x) \vee$
$(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v)) \vee$
$(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u)) \vee$
$(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
$(\forall\ u \in nset\ us.\ \neg\ P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
$) \wedge P\ x)$
using *assms NNil-eq-nfilterD*[*of nell P x*] **by** *simp*

lemma *nfilter-eq-NNil-iff*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $(nfilter\ P\ nell\ = (NNil\ x)) \longleftrightarrow$
$(\exists\ us\ vs\ .\ (nell = (NNil\ x) \vee$
$(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v)) \vee$
$(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u)) \vee$
$(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
$(\forall\ u \in nset\ us.\ \neg\ P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
$) \wedge P\ x)$
proof −
have *1*: $(nfilter\ P\ nell\ = (NNil\ x)) \Longrightarrow$
$(\exists\ us\ vs\ .\ (nell = (NNil\ x) \vee$
$(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v)) \vee$
$(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u)) \vee$
$(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
$(\forall\ u \in nset\ us.\ \neg\ P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
$) \wedge P\ x)$
using *assms nfilter-eq-NNilD*[*of nell P x*] **by** *simp*
have *2*: $(\exists\ us\ vs\ .\ (nell = (NNil\ x) \vee$
$(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v)) \vee$
$(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u)) \vee$
$(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
$(\forall\ u \in nset\ us.\ \neg\ P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
$) \wedge P\ x) \Longrightarrow (nfilter\ P\ nell\ = (NNil\ x))$
using *nfilter-NCons-a nfilter-NNil*[*of P x*]
by (*auto simp add*: *nfilter-nappend1*)

show *?thesis*
  using *1 2* **by** *blast*
qed

lemma *NCons-eq-nfilterD-help*:
assumes ¬ *lnull ys*
  ¬ *lnull xs*
  LCons x xs = *lfilter P ys*
  xa ∈ *lset ys*
  P xa
shows ∃ *us*. ¬ *lnull us* ∧
  (∃ *vs*. (∃ *x*∈*lset vs*. *P x*) ∧
    ((∃ *x*∈*lset vs*. *P x*) ⟶
     ¬ *lnull vs* ∧ (*ys* = *LCons x vs* ∨ *ys* = *lappend us* (*LCons x vs*) ∧ *lfinite us* ∧
      (∀ *u*∈*lset us*. ¬ *P u*)) ∧ *P x* ∧ *xs* = *lfilter P vs*))
unfolding *lnull-def* **using** *assms lfilter-eq-LConsD*[*of P ys x xs*]
by (*metis diverge-lfilter-LNil lappend-code*(*1*) *lfilter-LNil llist.disc*(*1*))

lemma *NCons-eq-nfilterD*:
assumes ∃ *x* ∈ *nset nell*. *P x*
  (*NCons x nell1*) = *nfilter P nell*
shows (∃ *us vs*.
  (*nell* = (*NCons x vs*) ∨
   nell = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∧
   (∃ *x* ∈ *nset vs*. *P x*) ∧ *P x* ∧ *nell1* = *nfilter P vs*)
using *assms* **by** *transfer* (*auto split*: *if-split-asm simp add*: *NCons-eq-nfilterD-help*)

lemma *nfilter-eq-NConsD*:
assumes ∃ *x* ∈ *nset nell*. *P x*
  nfilter P nell = (*NCons x nell1*)
shows (∃ *us vs*.
  (*nell* = (*NCons x vs*) ∨
   nell = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∧
   (∃ *x* ∈ *nset vs*. *P x*) ∧ *P x* ∧ *nell1* = *nfilter P vs*)
using *assms NCons-eq-nfilterD*[*of nell P x nell1*] **by** *simp*

lemma *nfilter-eq-NCons-iff*:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *nfilter P nell* = (*NCons x nell1*) ⟷
  (∃ *us vs*.
  (*nell* = (*NCons x vs*) ∨
   nell = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∧
   (∃ *x* ∈ *nset vs*. *P x*) ∧ *P x* ∧ *nell1* = *nfilter P vs*)
proof −
have *1*: *nfilter P nell* = (*NCons x nell1*) ⟹
  (∃ *us vs*.
  (*nell* = (*NCons x vs*) ∨
   nell = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∧
   (∃ *x* ∈ *nset vs*. *P x*) ∧ *P x* ∧ *nell1* = *nfilter P vs*)
  using *assms nfilter-eq-NConsD*[*of nell P x nell1*] **by** *simp*

have *2*: (∃ *us vs*.
(*nell* = (*NCons x vs*) ∨
nell = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∧
(∃ *x* ∈ *nset vs*. *P x*) ∧ *P x* ∧ *nell1* = *nfilter P vs*) ⟹ *nfilter P nell* = (*NCons x nell1*)
using *nfilter-nappend1*[*of - P*] *nfilter-NCons*[*of - P x*]
by (*auto*, *meson nfilter-NCons*, *metis*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *nfilter-id-conv*:
assumes (∃ *x* ∈ *nset nell*. *P x*)
shows (*nfilter P nell* = *nell*) = (∀ *x*∈*nset nell*. *P x*) (**is** *?lhs* = *?rhs*)
using *assms* **by** *transfer* (*auto simp add*: *lfilter-id-conv*)

lemma *nfilter-idem*:
assumes (∃ *x* ∈ *nset nell*. *P x*)
shows *nfilter P* (*nfilter P nell*) = *nfilter P nell*
using *assms* **by** *transfer auto*

lemma *nfilter-ndropn-nlength*:
assumes *k* ≤ *nlength nell*
∃ *x* ∈ *nset* ((*ndropn k nell*)). *P x*
shows *nlength*(*nfilter P* ((*ndropn k nell*))) ≤ *nlength* (*nfilter P* (*nell*))
using *assms*
by *transfer*
(*auto simp add*: *min-def dest*: *in-lset-ldropnD*,
metis co.enat.exhaust-sel epred-le-epredI iless-Suc-eq lfilter-ldropn-llength llength-eq-0)

1.2.20 Setup for Lifting/Transfer

Relator and predicator properties

abbreviation *nellist-all* == *pred-nellist*

Transfer rules for the Transfer package

context includes *lifting-syntax*
begin

lemma *set1-pre-nellist-transfer* [*transfer-rule*]:
(*rel-pre-nellist A C* ===> *rel-set A*) *set1-pre-nellist set1-pre-nellist*
by(*auto simp add*: *rel-pre-nellist-def vimage2p-def rel-fun-def set1-pre-nellist-def rel-set-def*
collect-def sum-set-defs prod-set-defs elim: *rel-sum.cases split*: *sum.split-asm*)

lemma *set2-pre-nellist-transfer* [*transfer-rule*]:
(*rel-pre-nellist A C* ===> *rel-set C*) *set2-pre-nellist set2-pre-nellist*
by (*auto simp add*: *rel-pre-nellist-def vimage2p-def rel-fun-def set2-pre-nellist-def*
rel-set-def collect-def sum-set-defs prod-set-defs elim: *rel-sum.cases split*: *sum.split-asm*)

lemma *NCons-transfer2* [*transfer-rule*]:
(*A* ===> *nellist-all2 A* ===> *nellist-all2 A*) *NCons NCons*

unfolding *rel-fun-def* **by** *simp*
declare *NCons-transfer* [*transfer-rule*]

lemma *case-nellist-transfer* [*transfer-rule*]:
((A ===> C) ===> (A ===> *nellist-all2* A ===> C) ===> *nellist-all2* A ===> C)
case-nellist case-nellist
unfolding *rel-fun-def*
by (*simp add*: *nellist-all2-NNil1 nellist-all2-NNil2 split*: *nellist.split*)

lemma *unfold-nellist-transfer* [*transfer-rule*]:
((A ===> (=)) ===> (A ===> C) ===> (A ===> C) ===> (A ===> A) ===> A ===> *nellist-all2* C)
unfold-nellist unfold-nellist
proof(*rule rel-funI*)+
fix *IS-NNIL1* :: $'a \Rightarrow$ *bool* **and** *IS-NNIL2*
NLAST1 NLAST2 NHD1 NHD2 NTL1 NTL2 x y
assume *rel*: (A ===> (=)) *IS-NNIL1 IS-NNIL2* (A ===> C) *NLAST1 NLAST2*
(A ===> C) *NHD1 NHD2* (A ===> A) *NTL1 NTL2*
and A *x y*
show *nellist-all2 C* (*unfold-nellist IS-NNIL1 NLAST1 NHD1 NTL1 x*)
(*unfold-nellist IS-NNIL2 NLAST2 NHD2 NTL2 y*)
using ‹A *x y*› **using** *rel* **by** (*coinduction arbitrary*: *x y*) (*auto 4 4 elim*: *rel-funE*)
qed

lemma *corec-nellist-transfer* [*transfer-rule*]:
((A ===> (=)) ===> (A ===> C) ===> (A ===> C) ===> (A ===> (=)) ===> (A ===> *nellist-all2* C)
===> (A ===> A) ===> A ===> *nellist-all2* C) *corec-nellist corec-nellist*
by (*simp add*: *nellist.corec-transfer*)

lemma *ntl-transfer2* [*transfer-rule*]:
(*nellist-all2* A ===> *nellist-all2* A) *ntl ntl*
unfolding *ntl-def*[*abs-def*] **by** *transfer-prover*
declare *ntl-transfer* [*transfer-rule*]

lemma *nset-transfer2* [*transfer-rule*]:
(*nellist-all2* A ===> *rel-set* A) *nset nset*
by (*simp add*: *nellist.set-transfer*)

lemma *nmap-transfer4* [*transfer-rule*]:
((A ===> B) ===> *nellist-all2* A ===> *nellist-all2* B) *nmap nmap*
by (*simp add*: *nellist.map-transfer*)
declare *nmap-transfer* [*transfer-rule*]

lemma *is-NNil-transfer2* [*transfer-rule*]:
(*nellist-all2* A ===> (=)) *is-NNil is-NNil*
by(*auto dest*: *nellist-all2-is-NNilD*)
declare *is-NNil-transfer* [*transfer-rule*]

lemma *snocn-transfer2* [*transfer-rule*]:

(*nellist-all2 A* ===> *A* ===> *nellist-all2 A*) *snocn snocn*
unfolding *rel-fun-def*
by (*metis nappend-snocn nellist-all2-NNil nellist-all2-nappendI*)
declare *snocn-transfer*[*transfer-rule*]

lemma *nappend-transfer* [*transfer-rule*]:
(*nellist-all2 A* ===> (*nellist-all2 A*) ===> *nellist-all2 A*) *nappend nappend*
by(*auto intro*: *nellist-all2-nappendI elim*: *rel-funE*)
declare *nappend.transfer* [*transfer-rule*]

lemma *lappendn-transfer* [*transfer-rule*]:
(*llist-all2 A* ===> *nellist-all2 A* ===> *nellist-all2 A*) *lappendn lappendn*
unfolding *rel-fun-def*
by *transfer*(*auto intro*: *llist-all2-lappendI*)
declare *lappendn.transfer* [*transfer-rule*]

lemma *nellist-of-llist-a-transfer2* [*transfer-rule*]:
(*A* ===> *llist-all2 A* ===> *nellist-all2 A*) *nellist-of-llist-a nellist-of-llist-a*
by (*simp add*: *rel-funI*)
declare *nellist-of-llist-a-transfer* [*transfer-rule*]

lemma *nlength-transfer* [*transfer-rule*]:
(*nellist-all2 A* ===> (=)) *nlength nlength*
by(*auto dest*: *nellist-all2-nlengthD*)
declare *nlength.transfer* [*transfer-rule*]

lemma *ndropn-transfer* [*transfer-rule*]:
((=) ===> *nellist-all2 A* ===> *nellist-all2 A*) *ndropn ndropn*
unfolding *rel-fun-def*
by *transfer* (*auto intro*: *llist-all2-ldropnI simp add*: *llist-all2-ldropnI llist-all2-llengthD*)
declare *ndropn.transfer* [*transfer-rule*]

lemma *ntake-transfer* [*transfer-rule*]:
((=) ===> *nellist-all2 A* ===> *nellist-all2 A*) *ntake ntake*
unfolding *rel-fun-def*
by *transfer* (*auto simp add*: *llist-all2-ltakeI*)
declare *ntake.transfer* [*transfer-rule*]

lemma *ntaken-transfer* [*transfer-rule*]:
((=) ===> *nellist-all2 A* ===> *nellist-all2 A*) *ntaken ntaken*
unfolding *rel-fun-def*
by *transfer* (*auto simp add*: *llist-all2-ltakeI*)
declare *ntaken.transfer* [*transfer-rule*]

lemma *nzip-transfer* [*transfer-rule*]:
(*nellist-all2 A* ===> *nellist-all2 B* ===> *nellist-all2* (*rel-prod A B*)) *nzip nzip*
by (*auto intro*: *nellist-all2-nzipI*)

lemma *niterates-transfer* [*transfer-rule*]:

```
  ((A ===> A) ===>  A ===> nellist-all2 A) niterates niterates
unfolding rel-fun-def
apply transfer
using iterates-transfer unfolding rel-fun-def
by blast

lemma nfilter-transfer [transfer-rule]:
  ( (A ===> (=)) ===> nellist-all2 A ===> nellist-all2 A) nfilter nfilter
unfolding rel-fun-def
by transfer
   (auto intro: llist-all2-lfilterI  dest: llist-all2-lfiniteD llist-all2-lsetD1,
    meson llist-all2-lsetD2)
declare nfilter.transfer [transfer-rule]

lemma nconcat-transfer [transfer-rule]:
  ( nellist-all2 (nellist-all2 A)  ===> nellist-all2 A ) nconcat nconcat
unfolding rel-fun-def
using nellist-all2-nconcatI
by auto
declare nconcat.transfer [transfer-rule]

lemma nellist-all2-rsp:
  assumes R1: ∀ x y. R1 x y ⟶ (∀ a b. R1 a b ⟶ S x a = T y b)
  and R2: ∀ x y. R2 x y ⟶ (∀ a b. R2 a b ⟶ S' x a = T' y b)
  and xsys: nellist-all2 R1 xs ys
  and xs'ys': nellist-all2 R1 xs' ys'
  shows nellist-all2 S xs xs' = nellist-all2 T ys ys'
proof
  assume nellist-all2 S xs xs'
  with xsys xs'ys' show nellist-all2 T ys ys'
  proof(coinduction arbitrary: ys ys' xs xs')
    case (ilist-all2 ys ys' xs xs')
    thus ?case
      by cases (auto 4 4 simp add: nellist-all2-NCons1 nellist-all2-NCons2 nellist-all2-NNil1
              nellist-all2-NNil2 dest: R1[rule-format] R2[rule-format])
  qed
next
  assume nellist-all2 T  ys ys'
  with xsys xs'ys' show nellist-all2 S xs xs'
  proof(coinduction arbitrary: xs xs' ys ys')
    case (ilist-all2 xs xs' ys ys')
    thus ?case
      by cases (auto 4 4 simp add: nellist-all2-NCons1 nellist-all2-NCons2 nellist-all2-NNil1
              nellist-all2-NNil2 dest: R1[rule-format] R2[rule-format])
  qed
qed

lemma nellist-all2-transfer2 [transfer-rule]:
  ((R1 ===> R1 ===> (=) ) ===>
    nellist-all2 R1 ===> nellist-all2 R1 ===> (=)) nellist-all2 nellist-all2
```

by (*simp add*: *nellist-all2-rsp rel-fun-def*)
declare *nellist-all2-transfer* [*transfer-rule*]

end

Delete lifting rules for *'a nellist* because the parametricity rules take precedence over most of the transfer rules. They can be restored by including the bundle *nellist.lifting*.

lifting-update *nellist.lifting*
lifting-forget *nellist.lifting*

end

1.3 Extra operations on non-empty conductive lists

The operations ndropns, nkfilter, nleast, nidx, nfuse, nbutlast, nsubn, nfuse, nridx, nlastnfirst, nfuse-cat, nsubn_nellist, lsum, add_zero, all_nfinite, all_gr_zero and nrev are defined for nellists together with a library of lemmas.

theory *NELList-Extras*
imports *NELList*
begin

1.3.1 ndropns

primcorec *ndropns* :: *'a nellist* ⇒ *'a nellist nellist*
where *ndropns nell* =
(*case nell of* (*NNil b*) ⇒ (*NNil* (*NNil b*)) |
(*NCons x nell'*) ⇒ (*NCons* (*NCons x nell'*) (*ndropns nell'*)))

simps-of-case *ndropns-code* [*code*, *simp*, *nitpick-simp*]: *ndropns.code*

lemma *ndropns-simps* [*simp*]:
shows *nhd-ndropns*: ¬ *is-NNil nell* ⟹ *nhd* (*ndropns nell*) = *nell*
and *ndropns-NCons*: *ntl* (*ndropns nell*) = (*case nell of* (*NNil b*) ⇒ (*NNil* (*NNil b*)) |
(*NCons x nell'*) ⇒ *ndropns nell'*)
by (*auto simp add*: *nellist.case-eq-if*)
(*metis ndropns-code*(*1*) *nellist.collapse*(*1*) *nellist.sel*(*4*))

lemma *ndropns-nnth*:
assumes *i* ≤ *nlength nell*
shows *nnth* (*ndropns nell*) *i* = *ndropn i nell*
using *assms*
proof (*induction i arbitrary*: *nell*)
case *0*
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** (*simp add*: *nnth-NNil*)
next
case (*NCons x21 x22*)

then show *?thesis* **by** *simp*
qed
next
case (*Suc i*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** (*simp add*: *nnth-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis* **using** *Suc* **by** (*simp add*: *Suc-ile-eq*)
qed
qed

lemma *ndropns-nlength*:
nlength (*ndropns nell*) = (*nlength nell*)
by (*coinduction arbitrary*: *nell rule*: *enat-coinduct*)
(*case-tac nell*, *auto*)

lemma *in-nset-ndropns*:
nell ∈ *nset*(*ndropns nellx*) ⟷ (∃ *i*. *i* ≤ *nlength nellx* ∧ *nell* = *ndropn i nellx*)
by (*metis in-nset-conv-nnth ndropns-nlength ndropns-nnth*)

lemma *nset-ndropns*:
nset (*ndropns nell*) = { *ndropn i nell* | *i*. *i* ≤ *nlength nell*}
using *in-nset-conv-nnth*[*of - ndropns nell*] *nset-conv-nnth*[*of ndropns nell*]
using *in-nset-ndropns*[*of - nell*] **by** *auto*

lemma *nmap-first-ndropns*:
nmap (λ *nell*. *nnth nell 0*) (*ndropns nell*) = *nell*
by (*coinduction arbitrary*: *nell*)
(*case-tac nell*, *auto simp add*: *nnth-NNil*)

lemma *ndropn-ndropns*:
assumes *i* ≤ *nlength*(*ndropns nell*)
shows *ndropn i* (*ndropns nell*) = *ndropns* (*ndropn i nell*)
using *assms*
proof (*coinduction arbitrary*: *nell i*)
case (*Eq-nellist ia nellx*)
then show *?case*
proof −
have *1*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
is-NNil (*ndropn nellx* (*ndropns ia*)) = *is-NNil* (*ndropns* (*ndropn nellx ia*))
by (*simp add*: *is-NNil-ndropn ndropns-nlength*)
have *2*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
(*is-NNil* (*ndropn nellx* (*ndropns ia*)) ⟶
is-NNil (*ndropns* (*ndropn nellx ia*)) ⟶
nlast (*ndropn nellx* (*ndropns ia*)) = *nlast* (*ndropns* (*ndropn nellx ia*)))
by (*metis dual-order.antisym ndropn-eq-NNil ndropns.disc*(*2*) *ndropns.simps*(*3*) *ndropns-nlength*
ndropns-nnth nellist.case-eq-if nellist.collapse(*1*) *the-enat.simps*)

have *3*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
(¬ *is-NNil* (*ndropn nellx* (*ndropns ia*)) ⟶
¬ *is-NNil* (*ndropns* (*ndropn nellx ia*)) ⟶
nhd (*ndropn nellx* (*ndropns ia*)) = *nhd* (*ndropns* (*ndropn nellx ia*)))
by (*metis Nat.add-0-right ndropn-nnth ndropns.disc*(*1*) *ndropns-nlength ndropns-nnth nhd-conv-nnth nhd-ndropns*)
have *4*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
(¬ *is-NNil* (*ndropn nellx* (*ndropns ia*)) ⟶
¬ *is-NNil* (*ndropns* (*ndropn nellx ia*)) ⟶
(∃ *nell i*.
ntl (*ndropn nellx* (*ndropns ia*)) = *ndropn i* (*ndropns nell*) ∧
ntl (*ndropns* (*ndropn nellx ia*)) = *ndropns* (*ndropn i nell*) ∧
enat i ≤ *nlength* (*ndropns nell*)))
by (*metis Suc-ile-eq is-NNil-ndropn ndropn-Suc-conv-ndropn ndropns-code*(*2*) *ndropns-nlength nellist.sel*(*5*) *order.order-iff-strict*)
show *?thesis*
using *1 2 3 4 Eq-nellist* **by** *blast*
qed
qed

lemma *ndropns-nfilter-nnth*:
assumes *i* ≤ *nlength* (*nfilter P* (*ndropns nell*))
∃ *nelly* ∈ *nset*(*ndropns nell*). *P nelly*
shows *P* (*nnth* (*nfilter P* (*ndropns nell*)) *i*)
using *assms* **using** *nset-nfilter*[*of ndropns nell P*]
by (*metis* (*full-types*) *Int-iff in-nset-conv-nnth mem-Collect-eq*)

lemma *nnth-zero-ndropn*:
nnth (*ndropn n nell*) *0* = *nnth nell n*
by *simp*

lemma *in-nset-ndropns-nhd*:
x ∈ *nset nell* ⟷(∃ *ys*. *x*=(*nnth ys 0*) ∧ *ys* ∈ *nset*(*ndropns nell*))
by *auto*
(*metis in-nset-conv-nnth ndropns-nlength nmap-first-ndropns nnth-nmap*,
metis Nat.add-0-right in-nset-conv-nnth in-nset-ndropns ndropn-nnth)

lemma *nset-ndropns-nhd*:
nset nell = {(*nnth nelly 0*) | *nelly*. *nelly* ∈ *nset*(*ndropns nell*) }
by *auto*
(*meson in-nset-ndropns-nhd*,
metis in-nset-conv-nnth in-nset-ndropns nnth-zero-ndropn)

lemma *nellist-all2-ndropnsI*:
nellist-all2 A nellx nelly ⟹ *nellist-all2* (*nellist-all2 A*) (*ndropns nellx*) (*ndropns nelly*)
by (*coinduction arbitrary*: *nellx nelly*)
(*auto simp add*: *nellist.case-eq-if dest*: *nellist-all2-nhdD nellist-all2-is-NNilD*
intro: *nellist-all2-ntlI*)

1.3.2 lsum

primcorec *lsum* :: *'a nellist nellist ⇒ nat ⇒ nat nellist*
where
lsum nells a = (case nells of (NNil nell) ⇒ (NNil (a+(the-enat (nlength nell)))) |
(NCons nell nells1) ⇒ (NCons (a+(the-enat (nlength nell)))
(lsum nells1 (a+(the-enat (nlength nell))))))

simps-of-case *lsum-code* [*code*, *simp*, *nitpick-simp*]: *lsum.code*

definition *addzero* :: *nat nellist ⇒ nat nellist*
where *addzero ls = (if nlength ls = 0 then*
(if nfirst ls = 0 then ls else (NCons 0 ls)) else (NCons 0 ls))

abbreviation *all-nfinite nells* ≡ (∀ *nell* ∈ *nset nells. nfinite nell*)

abbreviation *all-gr-zero nells* ≡ (∀ *nell* ∈ *nset nells. 0<nlength nell*)

1.3.3 Definitions

context
includes *nellist.lifting*
begin

lift-definition *nkfilter* :: *('a ⇒ bool) ⇒ nat ⇒'a nellist ⇒ nat nellist*
is *λ P n xs. (if lnull(kfilter P n xs) then (iterates Suc n) else kfilter P n xs)*
by *simp*

lift-definition *nleast* :: *('a ⇒ bool) ⇒ 'a nellist ⇒ nat*
is *λ P xs. lleast P xs*
by *auto*

lift-definition *nridx* :: *('a ⇒ 'a ⇒ bool) ⇒ 'a nellist ⇒ bool*
is *λ R xs. ridx R xs*
by *auto*

lift-definition *nidx* :: *nat nellist ⇒ bool*
is *λ xs. lidx xs*
by *auto*

lift-definition *nbutlast* :: *'a nellist ⇒ 'a nellist*
is *λ xs. (if lnull (lbutlast xs) then (LCons (llast xs) LNil) else lbutlast xs)*
by *auto*

lift-definition *nfuse* :: *'a nellist ⇒ 'a nellist ⇒ 'a nellist*
is *λ xs ys. lfuse xs ys*
using *lfuse-conv-lnull* **by** *blast*

lift-definition *nsubn* :: *'a nellist ⇒ nat ⇒ nat ⇒ 'a nellist*
is *(λ xs n1 n2. lsubc n1 n2 xs)*
unfolding *lsubc-def*

by *auto*
(*metis co.enat.exhaust-sel dual-order.refl enat-ile iless-Suc-eq leD llength-eq-0*)

lift-definition *nlastnfirst* :: *'a nellist nellist ⇒ bool*
is *λ xss. llastlfirst xss*
apply (*simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)
unfolding *llastlfirst-def*
using *llist-all2-llist-of-nellist-1* **by** *blast*

lemma *nfusecat-help*:
assumes *∃ y. llist-all2 (λx z. llist-of-nellist z = x) llist1 y ∧ y ≠ LNil ∧*
llist-all2 (λx z. llist-of-nellist z = x) llist2 y
shows *lfusecat llist1 ≠ LNil ∧ lfusecat llist1 = lfusecat llist2*
proof −
have *1*: *lfusecat llist1 = lfusecat llist2*
using *assms llist-all2-llist-of-nellist-1 lnull-def* **by** *blast*
have *2*: *lfusecat llist1 ≠ LNil*
using *assms* **apply** *auto*
using *lfusecat-not-lnull-var llist-all2-lnullD llist-all2-lsetD1* **by** *fastforce*
show *?thesis* **using** *1 2* **by** *auto*
qed

lift-definition *nfusecat* :: *'a nellist nellist ⇒ 'a nellist*
is *λ xss. lfusecat xss*
using *nfusecat-help*
by (*simp add*: *pcr-nellist-def OO-def cr-nellist-def nellist.pcr-cr-eq rel-fun-def lnull-def*
llist-all2-cases llist-all2-rsp llist.rel-eq not-lnull-conv-llist-of-nellist) *blast*

lift-definition *nrev* :: *'a nellist ⇒ 'a nellist*
is *λ xs. if lfinite xs then lrev xs else xs*
by *auto*

1.3.4 nbutlast

lemma *nbutlast-NNil*[*simp*]:
nbutlast (NNil x) = (NNil x)
apply *transfer*
by *auto*

lemma *nbutlast-snoc* [*simp*]:
nbutlast (nappend nell (NNil x)) = nell
apply *transfer*
by *auto*

lemma *nbutlast-def1*:
nbutlast nell =
(*case nell of (NNil x) ⇒ (NNil x) |*
(NCons x nell1) ⇒
(*case nell1 of (NNil y) ⇒ (NNil x) |*
(NCons z nell2) ⇒ (NCons x (nbutlast nell1))))

proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
  proof −
   have *1*: *is-NNil x22* ⟹ *nbutlast* (*NCons x21 x22*) = (*NNil x21*)
    by (*metis nappend-NNil nbutlast-snoc nellist.collapse*(*1*))
   have *2*: ¬*is-NNil x22* ⟹ *nbutlast* (*NCons x21 x22*) = (*NCons x21* (*nbutlast x22*))
    apply *transfer*
    by *auto*
      (*metis lhd-LCons-ltl llist.collapse*(*1*),
       metis lbutlast-simps(*3*) *lhd-LCons-ltl*)
   show *?thesis*
   by (*simp add*: *1 2 NCons nellist.case-eq-if*)
  qed
qed

lemma *ntl-nbutlast*:
  ntl (*nbutlast nell*) =
    (*if is-NNil nell then ntl nell else*
      (*if is-NNil* (*ntl nell*) *then NNil* (*nhd nell*) *else nbutlast* (*ntl nell*)))
by (*auto simp add*: *nbutlast-def1 nellist.case-eq-if*)

lemma *nbutlast-not-nfinite*:
 assumes ¬ *nfinite nell*
 shows *nbutlast nell* = *nell*
 using *assms*
apply *transfer*
by *simp*
  (*metis lbutlast.disc-iff*(*2*) *lbutlast-not-lfinite*)

lemma *nbutlast-nfinite*:
 nfinite (*nbutlast nell*) ⟷ *nfinite nell*
apply *transfer*
by (*metis* (*full-types*) *lbutlast-lfinite lbutlast-not-lfinite lfinite-LNil lfinite-code*(*2*))

lemma *nlength-nbutlast* [*simp*]:
  nlength (*nbutlast nell*) = *epred* (*nlength nell*)
apply *transfer*
by (*simp*, *simp add*: *epred-llength*)

lemma *nbutlast-nappend*:
  nbutlast (*nappend nellx nelly*) =
    (*if is-NNil nelly then nellx else nappend nellx* (*nbutlast nelly*))
apply *transfer*
by *simp*
  (*metis lbutlast-lappend lbutlast-snoc lhd-LCons-ltl lnull-def*)

lemma *nappend-nbutlast-nlast-id-nfinite*:
assumes *nfinite nellx*
¬*is-NNil nellx*
shows (*nappend* (*nbutlast nellx*) (*NNil* (*nlast nellx*))) = *nellx*
using *assms*
apply *transfer*
by *simp*
(*metis lhd-LCons-ltl llist.collapse*(*1*))

lemma *nappend-nbutlast-nlast-id-not-nfinite*:
assumes ¬*nfinite nellx*
¬ *is-NNil nellx*
shows (*nappend* (*nbutlast nellx*) (*NNil* (*nlast nellx*))) = *nellx*
using *assms*
apply *transfer*
by *simp*
(*metis lappend-inf lbutlast.disc-iff*(*2*) *lbutlast-snoc*)

lemma *nappend-nbutlast-nlast-id* [*simp*]:
shows ¬ *is-NNil nell* ⟹ (*nappend* (*nbutlast nell*) (*NNil* (*nlast nell*))) = *nell*
using *nappend-nbutlast-nlast-id-nfinite nappend-nbutlast-nlast-id-not-nfinite* **by** *blast*

lemma *nbutlast-eq-NNil-conv*:
nbutlast nell = (*NNil* (*nfirst nell*)) ⟷
nell= (*NNil* (*nfirst nell*)) ∨ (∃ *x*. *nell* = (*NCons x* (*NNil* (*nlast nell*))))
apply *transfer*
by (*auto simp add*: *llist.expand lbutlast-not-lfinite*)
(*metis lhd-LCons-ltl llast-LCons*,
metis eq-LConsD lbutlast-eq-LNil-conv lbutlast-ltl lhd-LCons-ltl llast-LCons2 llast-singleton,
simp add: *eq-LConsD*,
simp add: *eq-LConsD lbutlast-eq-LCons-conv*,
metis lfinite-LNil lfinite-ltl llist.collapse(*1*))

lemma *nbutlast-eq-NCons-conv*:
nbutlast nell = (*NCons x ys*) ⟷
nell = (*NCons x* (*nappend ys* (*NNil* (*nlast nell*))))
apply *transfer*
by (*auto simp add*: *eq-LConsD lbutlast-eq-LCons-conv llast-linfinite*)

lemma *nbutlast-conv-ntake*:
nbutlast nell = *ntake* (*epred* (*nlength nell*)) *nell*
apply *transfer*
by *simp*
(*metis co.enat.exhaust-sel ileI1 iless-eSuc0 lappend-lbutlast-llast-id lappend-lnull1*
lbutlast.disc-iff(*2*) *lbutlast-conv-ltake llength-eq-0 llength-lbutlast ltake-all*)

lemma *nmap-nbutlast*:
nmap f (*nbutlast nell*) = *nbutlast* (*nmap f nell*)
apply *transfer*

by *simp*
(*metis lfinite-ltl llast-lmap lmap-lbutlast lnull-imp-lfinite*)

lemma *snocs-eq-iff-nbutlast*:
nappend nell (*NNil x*) = *nell1* ⟷
((*nfinite nell1* ∧ ¬ *is-NNil nell1* ∧ *nbutlast nell1* = *nell* ∧ *nlast nell1* = *x*)
∨ (¬ *nfinite nell1* ∧ *nbutlast nell* = *nell1*))
by (*metis is-NNil-nappend nappend-inf nappend-nbutlast-nlast-id nbutlast-nfinite nbutlast-snoc nlast-NNil nlast-nappend*)

lemma *in-nset-nbutlastD*:
x ∈ *nset*(*nbutlast nell*) ⟹ *x* ∈ *nset nell*
by (*metis in-nset-snocn-iff nappend-nbutlast-nlast-id-nfinite nappend-snocn nbutlast-NNil nbutlast-not-nfinite nellist.collapse*(*1*))

lemma *in-nset-nbutlast-nappendI*:
x ∈ *nset* (*nbutlast nell*) ∨ (*nfinite nell* ∧ ¬ *is-NNil nell1* ∧ *x* ∈ *nset*(*nbutlast nell1*)) ⟹
x ∈ *nset* (*nbutlast* (*nappend nell nell1*))
unfolding *nbutlast-nappend*
by (*metis* (*full-types*) *Un-iff in-nset-nbutlastD nset-nappend*)

lemma *nnth-nbutlast*:
assumes *n* ≤ *nlength*(*nbutlast nell*)
shows *nnth* (*nbutlast nell*) *n* = *nnth nell n*
by (*metis assms nappend-nbutlast-nlast-id-nfinite nbutlast-eq-NNil-conv nbutlast-not-nfinite ndropn-is-NNil ndropn-nfirst nellist.collapse*(*1*) *nnth-nappend1 nnth-nlast*)

1.3.5 nsubn

lemma *nsubn-def1*:
nsubn nell i j = *ntaken* (*j*−*i*) (*ndropn i nell*)
apply *transfer*
unfolding *lsubc-def*
by *auto*
(*metis co.enat.exhaust-sel dual-order.refl enat-ile iless-Suc-eq leD llength-eq-0* ,
metis eSuc-enat enat-the-enat infinity-ileE ldrop-enat min.cobounded1)

lemma *nsubn-same*:
shows *nsubn nell k k* = (*NNil* (*nnth nell k*))
unfolding *nsubn-def1*
by (*simp add*: *ndropn-nfirst*)

lemma *nsubn-nlength*:
nlength(*nsubn nell i j*) = *min* (*j*−*i*) (*nlength nell* − *i*)
by (*simp add*: *nsubn-def1*)

lemma *nsubn-nlength-gr-one*:
assumes *k*<*n*

$n \leq$ *nlength nell*
shows *0<* *nlength* (*nsubn nell k n*)
using *assms*
unfolding *nsubn-nlength*
by (*metis enat-minus-mono1 enat-ord-simps*(*2*) *idiff-enat-enat min-def zero-enat-def zero-less-diff*)

lemma *nsubn-nfinite*:
shows *nfinite* (*nsubn nell k n*)
by (*simp add*: *nsubn-def1*)

lemma *nsubn-nnth*:
shows *nnth* (*nsubn nell i j*) *k* = *nnth nell* (*i* + *min k* (*j* − *i*))
unfolding *nsubn-def1*
using *ntaken-nnth*[*of j−i* (*ndropn i nell*) *k*] *ndropn-nnth*[*of i nell* (*min k* (*j* − *i*))]
by *simp*

lemma *ntaken-ndropn*:
ntaken n (*ndropn k nell*) = *nsubn nell k* (*n*+*k*)
by (*simp add*: *nsubn-def1*)

lemma *ntaken-ndropn-nfirst*:
nfirst (*ntaken n* (*ndropn k nell*)) = *nnth nell k*
by (*metis min-0L ndropn-nfirst ntaken-0 ntaken-ntaken nlast-NNil*)

lemma *ntaken-ndropn-nfirst-a*:
nfirst (*ntaken n* (*ndropn k nell*)) = *nfirst*(*ndropn k nell*)
by (*simp add*: *ndropn-nfirst ntaken-ndropn-nfirst*)

lemma *ntaken-ndropn-nlast*:
nlast(*ntaken n* (*ndropn k nell*)) = *nnth nell* (*n*+*k*)
by (*simp add*: *add.commute ntaken-nlast*)

lemma *nsubn-nfirst*:
nfirst (*nsubn nell i j*) = *nnth nell i*
by (*simp add*: *nsubn-def1 ntaken-ndropn-nfirst*)

lemma *nsubn-nlast*:
nlast (*nsubn nell i j*) = *nnth nell* (*j* − *i* + *i*)
by (*simp add*: *nsubn-def1 ntaken-ndropn-nlast*)

lemma *nsubn-ndropn*:
assumes *i<j*
shows *nsubn nell* (*i*+*k*) (*j*+*k*) = *nsubn* (*ndropn k nell*) *i j*
using *assms*
by (*simp add*: *add.commute ndropn-ndropn nsubn-def1*)

lemma *pref-ntaken-3*:

(*ntaken i* (*ntaken* (*i*+*k*) *nell*)) = (*ntaken i nell*)
by (*metis le-add1 min.orderE ntaken-ntaken*)

lemma *ntaken-ndropn-swap-nlength*:
assumes *ia*+ *i* ≤ *nlength nell*
shows *nlength* (*ntaken ia* (*ndropn i nell*)) = *nlength* (*ndropn i* (*ntaken* (*ia*+*i*) *nell*))
using *assms ndropn-nlength*[*of i* (*ntaken* (*ia*+*i*) *nell*)] *ntaken-nlength*[*of ia* (*ndropn i nell*)]
by *auto*
(*metis add-diff-cancel-right*′ *enat-minus-mono1 idiff-enat-enat min.orderE*)

lemma *ntaken-ndropn-swap-nnth*:
assumes *m* ≤ *ia*
ia+ *i* ≤ *nlength nell*
shows *nnth* (*ntaken ia* (*ndropn i nell*)) *m* = *nnth* (*ndropn i* (*ntaken* (*ia*+*i*) *nell*)) *m*
using *assms*
by (*simp add*: *ntaken-nnth*)

lemma *nellist-eq-nnth-eq*:
(*nellx* = *nelly*) ⟷ *nlength nellx* = *nlength nelly* ∧ (∀ *i* ≤ *nlength nellx*. *nnth nellx i* = *nnth nelly i*)
by *transfer*
(*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llist-eq-lnth-eq min-absorb1 the-enat.simps*)

lemma *ntaken-ndropn-swap*:
assumes *ia*+ *i* ≤ *nlength nell*
shows (*ntaken ia* (*ndropn i nell*)) = (*ndropn i* (*ntaken* (*ia*+*i*) *nell*))
using *assms nellist-eq-nnth-eq*[*of* (*ntaken ia* (*ndropn i nell*)) (*ndropn i* (*ntaken* (*ia*+*i*) *nell*))]
using *ntaken-ndropn-swap-nlength*
using *ntaken-ndropn-swap-nnth* **by** *fastforce*

lemma *ntaken-nsubn*:
assumes *n* ≤ *nlength nell*
m + *k* ≤ *n*
shows *ntaken m* (*nsubn nell k n*) = *nsubn nell k* (*m*+*k*)
using *assms*
unfolding *nsubn-def1*
by *simp*

lemma *ndropn-nsubn*:
assumes *n* ≤ *nlength nell*
m + *k* ≤ *n*
shows *ndropn m* (*nsubn nell k n*) = *nsubn nell* (*m*+*k*) *n*
proof −
have *1*: *ntaken* (*n*−*k*) (*ndropn k nell*) = *ndropn k* (*ntaken n nell*)
by (*metis add-leD2 assms*(*1*) *assms*(*2*) *diff-add ntaken-ndropn-swap plus-enat-simps*(*1*))
have *2*: *ndropn m* (*ndropn k* (*ntaken n nell*)) =
ndropn (*m* +*k*) (*ntaken n nell*)
by (*simp add*: *add.commute ndropn-ndropn*)
show *?thesis* **unfolding** *nsubn-def1* **using** *1 2*
by (*simp add*: *assms*(*1*) *assms*(*2*) *ntaken-ndropn-swap*)

qed

lemma *ntl-nsubn*:
assumes $n \leq$ *nlength nell*
$k \leq n$
shows *ntl*(*nsubn nell k n*) = (*if n=k then* (*NNil* (*nnth nell k*)) *else nsubn* (*ntl nell*) *k* (*n−1*))
using *assms* **unfolding** *nsubn-def1*
using *ntl-ntaken*[*of n−k ndropn k nell*] *ntl-ndropn*[*of k nell*]
by (*metis diff-diff-cancel diff-right-commute diff-zero nellist.sel*(*4*) *nsubn-def1 nsubn-same*)

lemma *nsubn-nsubn*:
assumes *n1* ≤ *n2*
n0 ≤ *n4*
n2 ≤ *n4−n0*
n4 ≤ *n3*
n3 ≤ *nlength nell*
shows (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) = (*nsubn* (*nsubn nell n0 n4*) *n1 n2*)
proof −
have *1*: *nlength*(*nsubn nell n0 n3*) = *n3 −n0*
using *assms* **by** (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength*)
have *2*: *nlength* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) = *n2 −n1*
using *assms*
by (*metis Nat.le-diff-conv2 enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat le-trans min.orderE nsubn-nlength*)
have *3*: *nlength*(*nsubn nell n0 n4*) = *n4 −n0*
using *assms nsubn-nlength*[*of nell n0 n4*]
unfolding *min-def*
by (*metis enat-minus-mono1 idiff-enat-enat min.coboundedI1 min.left-commute min-absorb1 min-enat-simps*(*1*))
have *4*: *nlength* (*nsubn* (*nsubn nell n0 n4*) *n1 n2*) = *n2 −n1*
using *assms*
by (*metis 3 enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat min.orderE nsubn-nlength*)
have *5*: ⋀*i*. *i*≤ (*n3 −n0*) ⟶ (*nnth* (*nsubn nell n0 n3*) *i*) = (*nnth nell* (*n0+i*))
using *assms* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *6*: ⋀*i* . *i*≤ (*n2−n1*) ⟶
(*nnth* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) *i*) = (*nnth* (*nsubn nell n0 n3*) (*n1+i*))
using *assms* **by** (*simp add*: *Nat.le-diff-conv2 nsubn-nnth*)
have *7*: ⋀*i* . *i*≤ (*n2−n1*) ⟶
(*nnth* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) *i*) = (*nnth nell* (*n0+*(*n1+i*)))
using *5 6 assms* **by** *auto*
have *8*: *n0*≤ *n4* ∧ *n4* ≤ *nlength nell*
using *assms* **by** (*simp add*: *order-subst2*)
have *9*: ⋀*i*. *i*≤ (*n4 −n0*) ⟶ (*nnth* (*nsubn nell n0 n4*) *i*) = (*nnth nell* (*n0+i*))
using *8* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *10*: ⋀*i*. *i*≤ (*n2−n1*) ⟶
(*nnth* (*nsubn* (*nsubn nell n0 n4*) *n1 n2*) *i*) = (*nnth* (*nsubn nell n0 n4*) (*n1+i*))
by (*simp add*: *nsubn-def1 ntaken-nnth*)
have *11*: ⋀*i*. *i*≤ (*n2−n1*) ⟶
(*nnth* (*nsubn* (*nsubn nell n0 n4*) *n1 n2*) *i*) = (*nnth nell* (*n0 +*(*n1+i*)))
by (*metis 10 9 Nat.le-diff-conv2 add.commute assms*(*1*) *assms*(*3*) *le-trans*)

have *12*: $\bigwedge i.\ i \leq (n2-n1) \longrightarrow$
$(nnth\ (nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2\ )\ i) = (nnth\ (nsubn\ (nsubn\ nell\ n0\ n4)\ \ n1\ n2\ )\ i)$
by (*simp add*: *11 7*)
from *12 2 4* **show** *?thesis*
using *nellist-eq-nnth-eq*[*of* $(nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2\ )\ (nsubn\ (nsubn\ nell\ n0\ n4)\ \ n1\ n2\ )$]
enat-ord-simps(*1*) **by** *presburger*
qed

lemma *nsubn-nsubn-1*:
assumes $n1 \leq n2$
$n0 \leq n3$
$n2 \leq n3-n0$
$n3 \leq nlength\ nell$
shows $(nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2\ ) = (nsubn\ nell\ (n0+n1)\ (n0+n2))$
proof −
have *0*: $nlength(nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2\ ) = n2 - n1$
using *assms*
by (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength of-nat-eq-enat of-nat-mono*)
have *1*: $nlength(nsubn\ nell\ (n1+n0)\ (n2+n0)) = (n2+n0)\ -(n1+n0)$
using *assms nsubn-nlength*[*of nell* $(n1+n0)\ (n2+n0)$]
unfolding *min-def*
by (*metis Nat.le-diff-conv2 dual-order.trans enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat*)
have *10*: $(n2+n0)\ -(n1+n0) = n2\ -n1$
using *diff-cancel2* **by** *blast*
have *2*: $\bigwedge i.\ i \leq (n2-n1) \longrightarrow$
$(nnth\ (nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2\ )\ i) = (nnth\ nell\ (n0+(n1+i)))$
using *assms* **by** (*simp add*: *nsubn-nnth*)
have *3*: $\bigwedge i.\ i \leq (n2-n1) \longrightarrow$
$(nnth\ (nsubn\ nell\ (n0+n1)\ (n0+n2))\ i) = (nnth\ nell\ (n0+(n1+i)))$
using *assms* **by** (*metis 10 add.assoc add.commute min.orderE nsubn-nnth*)
have *4*: $\bigwedge i.\ i \leq (n2-n1) \longrightarrow$
$(nnth\ (nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2\ )\ i) = (nnth\ (nsubn\ nell\ (n0+n1)\ (n0+n2))\ i)$
by (*simp add*: *2 3*)
show *?thesis* **using** *nellist-eq-nnth-eq*[*of* $(nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2\ )$
$(nsubn\ nell\ (n0+n1)\ (n0+n2))$]
0 10 4 1
by (*metis add.commute enat-ord-simps*(*1*))
qed

lemma *nsubn-eq*:
assumes $n \leq m$
$m \leq nlength\ nellx$
$m \leq nlength\ nelly$
$\forall j.\ n \leq j \wedge j \leq m \longrightarrow nnth\ nellx\ j = nnth\ nelly\ j$
shows $nsubn\ nellx\ n\ m = nsubn\ nelly\ n\ m$
proof −
have *1*: $nlength\ (nsubn\ nellx\ n\ m) = nlength\ (nsubn\ nelly\ n\ m)$
using *assms* **by** (*metis enat-minus-mono1 min-def nsubn-nlength*)
have *2*: $\bigwedge j.\ j \leq nlength\ (nsubn\ nellx\ n\ m) \longrightarrow nnth\ (nsubn\ nellx\ n\ m)\ j = nnth\ (nsubn\ nelly\ n\ m)\ j$
using *assms* **by** (*simp add*: *Nat.le-diff-conv2 nsubn-nlength nsubn-nnth*)

show *?thesis*
by (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

1.3.6 nfuse

lemma *nfuse-def1*:
nfuse nellx nelly = (*if is-NNil nelly then nellx else nappend nellx* (*ntl nelly*))
apply *transfer*
unfolding *lfuse-def* **by** *simp force*

lemma *nfuse-NCons-a*:
nfuse (*NCons x nellx*) *nelly* = (*NCons x* (*nfuse nellx nelly*))
by (*simp add*: *nfuse-def1*)

lemma *nfuse-NCons-b*:
nfuse nellx (*NCons y nelly*) = *nappend nellx nelly*
by (*simp add*: *nfuse-def1*)

lemma *nfuse-simps* [*simp*]:
shows *nhd-nfuse*: *nhd*(*nfuse nellx nelly*) =
(*if is-NNil nellx then*
(*if is-NNil nelly then nhd nellx else nlast nellx*)
else nhd nellx)

and *ntl-nfuse*: *ntl*(*nfuse nellx nelly*) =
(*if is-NNil nellx then*
(*if is-NNil nelly then ntl nellx else ntl nelly*)
else (*if is-NNil nelly then ntl nellx*
else nappend (*ntl nellx*) (*ntl nelly*)))
by (*simp-all add*: *nfuse-def1*)

lemma *nfuse-nbutlast*:
assumes *nlast nellx* = *nfirst nelly*
¬ *is-NNil nellx*
¬ *is-NNil nelly*
shows *nfuse nellx nelly* = *nappend* (*nbutlast nellx*) *nelly*
using *assms*
by (*metis nappend-NCons nappend-NNil nappend-assoc nappend-nbutlast-nlast-id nappend-snocn*
nellist.collapse(*2*) *nellist.sel*(*3*) *nfuse-def1 nhd-snocn*)

lemma *nfuse-nlength*:
shows *nlength* (*nfuse nellx nelly*) = (*nlength nellx*) + (*nlength nelly*)
unfolding *nfuse-def1*
by(*cases nelly*) (*simp*, *simp add*: *eSuc-plus-1*)

lemma *nfuse-nnth*:
assumes *i*≤ *nlength* (*nfuse nellx nelly*)
nlast nellx = *nfirst nelly*
shows (*i* ≤ *nlength nellx* ⟶ *nnth* (*nfuse nellx nelly*) *i* = *nnth nellx i*)

```
        ∧
        (nlength nellx < i ∧ i≤ nlength (nfuse nellx nelly) ⟶
          nnth (nfuse nellx nelly) i = nnth nelly (i − (the-enat (nlength nellx))))
unfolding nfuse-def1
by (cases nelly)
   (auto simp add: nnth-nappend,
    metis Suc-diff-Suc  enat-iless enat-ord-simps(2) ndropn-Suc-NCons ndropn-nfirst the-enat.simps)

lemma nfuse-nnth-a:
 assumes  j ≤ nlength nelly
          nlast nellx = nfirst nelly
          nfinite nellx
 shows    nnth (nfuse nellx nelly) ((the-enat(nlength nellx)) +j) = (nnth nelly j)
 using assms unfolding nfuse-def1
 by (cases is-NNil nelly)
    (simp-all,
     metis assms(2) assms(3) is-NNil-def is-NNil-imp-nfinite ndropn-nlast ndropn-nfirst
     ndropn-nnth nnth-NNil,
     metis enat-le-plus-same(2) gen-nlength-def nappend-NNil ndropn-nlast ndropn-nappend2
     ndropn-nnth nellist.case-eq-if nellist.collapse(2) nfinite-nlength-enat nfirst-def
     nlength-code the-enat.simps)

lemma nfuse-nappend:
 assumes  nlast nellx = nfirst nelly
 shows    nfuse nellx nelly =
          (if nfinite nellx then
            (if is-NNil nellx then nelly else nappend (ntaken (the-enat(epred(nlength nellx))) nellx) nelly)
           else nellx)
proof (cases nelly)
 case (NNil x1)
 then show ?thesis
 proof (cases nfinite nellx)
 case True
 then show ?thesis
 proof (cases nellx)
  case (NNil x1)
  then show ?thesis using assms
   by (simp add: nfuse-def1)
      (metis nappend-snocn nellist.collapse(1) nellist.collapse(2) nhd-nappend nhd-snocn)
  next
  case (NCons x21 x22)
  then show ?thesis  unfolding nfuse-def1 using assms NNil NCons True
   by (auto simp add: nfirst-def)
      (metis True add-diff-cancel-left' assms eSuc-enat ndropn-nfirst ndropn-nlast
       nellist-split-2-last nfinite-nlength-enat nlast-NCons nlength-NCons plus-1-eq-Suc
       the-enat.simps zero-less-Suc)
  qed
  next
 case False
 then show ?thesis
```

by (*simp add*: *NNil nfuse-def1*)
qed
next
case (*NCons x21 x22*)
then show *?thesis*
proof (*cases nfinite nellx*)
case *True*
then show *?thesis*
unfolding *nfuse-def1*
proof *auto*
show *is-NNil nelly* $\Longrightarrow$ *is-NNil nellx* $\Longrightarrow$ *nellx* = *nelly*
by (*simp add*: *NCons*)
show *nfinite nellx* $\Longrightarrow$ *is-NNil nelly* $\Longrightarrow$ $\neg$ *is-NNil nellx* $\Longrightarrow$
nellx = *nappend* (*ntaken* (*the-enat* (*epred* (*nlength nellx*))) *nellx*) *nelly*
using *assms NCons True* **by** *simp*
show $\neg$ *is-NNil nelly* $\Longrightarrow$ *is-NNil nellx* $\Longrightarrow$ *nappend nellx* (*ntl nelly*) = *nelly*
using *assms NCons True*
by (*metis nappend-NNil nellist.collapse*(*1*) *nellist.sel*(*5*) *nnth-0 ntaken-0 ntaken-nlast*)
show *nfinite nellx* $\Longrightarrow$ $\neg$ *is-NNil nelly* $\Longrightarrow$ $\neg$ *is-NNil nellx* $\Longrightarrow$
nappend nellx (*ntl nelly*) = *nappend* (*ntaken* (*the-enat* (*epred* (*nlength nellx*))) *nellx*) *nelly*
using *assms NCons True*
by (*cases nellx*)
(*auto*,
metis True add-diff-cancel-left′ eSuc-enat nappend-NCons nappend-NNil nappend-assoc
ndropn-eq-NNil ndropn-nlast nellist.sel(*5*) *nellist-split-2-last nfinite-nlength-enat*
nlast-NNil nlast-ntl nlength-NCons nnth-0 ntaken-nlast ntl-ntaken-0 plus-1-eq-Suc
the-enat.simps zero-less-Suc)
qed
next
case *False*
then show *?thesis*
by (*simp add*: *nappend-inf nfuse-def1*)
qed
qed

lemma *nfuse-leftneutral* :
nfuse (*NNil* (*nfirst nell*)) *nell* = *nell*
by (*simp add*: *nfuse-nappend*)

lemma *nfuse-rightneutral* :
nfuse nell (*NNil* (*nlast nell*)) = *nell*
unfolding *nfuse-def*
by *simp*

lemma *nfirst-nfuse* :
assumes *nlast nellx* = *nfirst nelly*
shows *nfirst* (*nfuse nellx nelly*) = *nfirst nellx*
using *assms* **unfolding** *nfuse-def1* **by** *transfer auto*

lemma *nlast-nfuse* :

assumes *nlast nellx = nfirst nelly*
nfinite nellx
shows *nlast (nfuse nellx nelly) = nlast nelly*
using *assms* **unfolding** *nfuse-def1* **by** *transfer* (*auto*, *metis lhd-LCons-ltl llast-LCons llast-lappend*)

lemma *nfuseassoc* :
shows *(nfuse nellx (nfuse nelly nellz)) = (nfuse (nfuse nellx nelly) nellz)*
unfolding *nfuse-def1*
by *transfer* (*auto simp add*: *lappend-assoc*, *simp add*: *lappend-ltl*)

lemma *ntaken-nfuse* :
assumes *nlast nellx = nfirst nelly*
nfinite nellx
shows *(ntaken (the-enat (nlength nellx)) (nfuse nellx nelly)) = nellx*
proof (*cases is-NNil nelly*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-eq-NNil ndropn-nlast nfuse-def1 ntaken-all*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis add.left-neutral enat-le-plus-same(2) nfinite-nlength-enat nfuse-def1 ntaken-all*
ntaken-nappend1 the-enat.simps)
qed

lemma *ndropn-nfuse* :
assumes *nlast nellx = nfirst nelly*
nfinite nellx
shows *(ndropn (the-enat (nlength nellx)) (nfuse nellx nelly)) = nelly*
proof (*cases is-NNil nelly*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-nlast nfuse-def1 nfuse-leftneutral*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis enat-le-plus-same(2) gen-nlength-def ndropn-nappend2 ndropn-nlast nfinite-nlength-enat*
nfuse-def1 nfuse-leftneutral nlength-code the-enat.simps)
qed

lemma *nfuse-ntaken-ndropn-nlength* :
assumes *n ≤ nlength nell*
shows *nlength (nfuse (ntaken n nell) (ndropn n nell)) = nlength nell*
using *assms*
by (*metis dual-order.order-iff-strict enat-add-sub-same infinity-ileE less-eqE min.absorb1*
ndropn-nlength nfuse-nlength ntaken-nlength)

lemma *nfuse-ntaken-ndropn-nnth* :
assumes *n ≤ nlength nell*
i ≤ nlength nell
shows *nnth (nfuse (ntaken n nell) (ndropn n nell)) i = nnth nell i*
using *assms*

nfuse-nnth[*of i* (*ntaken n nell*) (*ndropn n nell*)]
nfuse-ntaken-ndropn-nlength[*of n nell*]
by (*metis dual-order.strict-iff-order enat-ord-simps*(*1*) *le-add-diff-inverse min.orderE ndropn-nfirst ndropn-nnth not-less ntaken-nlast ntaken-nlength ntaken-nnth the-enat.simps*)

lemma *nfuse-ntaken-ndropn*:
assumes *n*≤ *nlength nell*
shows *nfuse* (*ntaken n nell*) (*ndropn n nell*) = *nell*
using *assms*
by (*simp add*: *nfuse-ntaken-ndropn-nlength nfuse-ntaken-ndropn-nnth nellist-eq-nnth-eq*)

lemma *nfuse-nnth-var*:
assumes *enat i* ≤ *nlength* (*nfuse nellx nelly*)
nlast nellx = *nfirst nelly*
shows (*enat i* ≤ *nlength nellx* ⟶ *nnth* (*nfuse nellx nelly*) *i* = *nnth nellx i*) ∧
(*nlength nellx* ≤ *enat i* ∧ *enat i* ≤ *nlength* (*nfuse nellx nelly*) ⟶
nnth (*nfuse nellx nelly*) *i* = *nnth nelly* (*i* − *the-enat* (*nlength nellx*)))
using *nfuse-nnth assms*
by (*metis cancel-comm-monoid-add-class.diff-cancel dual-order.strict-iff-order ndropn-nfuse nlength-eq-enat-nfiniteD nnth-zero-ndropn the-enat.simps*)

lemma *nset-nfuse*:
nset (*nfuse nellx nelly*) =
(*if nfinite nellx then*
(*if is-NNil nelly then nset nellx else nset nellx* ∪ *nset* (*ntl nelly*))
else nset nellx)
by (*simp add*: *nfuse-def1 nset-nappend*)

lemma *nsubn-nfuse*:
assumes (*enat k*) ≤ *n*
(*enat n*) ≤ *m*
(*enat m*) ≤ *nlength nell*
shows *nfuse* (*nsubn nell k n*) (*nsubn nell n m*) = (*nsubn nell k m*)
using *assms*
proof −
have *1*: *nlast*(*nsubn nell k n*) = (*nnth nell n*)
by (*metis assms*(*1*) *enat-ord-simps*(*1*) *le-add-diff-inverse2 nsubn-def1 ntaken-ndropn-nlast*)
have *2*: *nfirst*(*nsubn nell n m*) = (*nnth nell n*)
by (*simp add*: *ndropn-nfirst nsubn-def1 ntaken-ndropn-nfirst-a*)
have *3*: *nlength* (*nsubn nell k n*) = *n* −*k*
using *assms nsubn-nlength*[*of nell k n*]
by (*metis dual-order.trans enat-minus-mono1 idiff-enat-enat min-absorb1*)
have *4*: *nlength* (*nsubn nell n m*) = *m* −*n*
by (*metis assms*(*3*) *enat-minus-mono1 idiff-enat-enat min-def nsubn-nlength*)
have *5*: *nlength* (*nsubn nell k m*) = *m* −*k*
by (*metis assms*(*3*) *enat-minus-mono1 idiff-enat-enat min-absorb1 nsubn-nlength*)
have *6*: *nlength*(*nfuse* (*nsubn nell k n*) (*nsubn nell n m*)) = *nlength*(*nsubn nell k m*)
unfolding *nsubn-def*
by (*metis 3 4 5 Nat.add-diff-assoc2 assms*(*1*) *assms*(*2*) *enat-ord-simps*(*1*) *nfuse-nlength nsubn-def ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-enat-simps*(*1*))

```
 have 7: (∀ i. i ≤ nlength(nsubn nell k m) ⟶
           (nnth (nfuse (nsubn nell k n) (nsubn nell n m)) i) = (nnth nell (k+i)))
  proof
    fix i
    show i ≤ nlength(nsubn nell k m) ⟶
           (nnth (nfuse (nsubn nell k n) (nsubn nell n m)) i) = (nnth nell (k+i))
    proof −
      have 41: nlength (nsubn nell k m) = (m−k)
        using 5 by blast
      have 42: i ≤ nlength (nsubn nell k m) ⟶ nnth (nfuse (nsubn nell k n) (nsubn nell n m)) i =
                 (if i ≤ n −k then (nnth (nsubn nell k n) i)
                                  else (nnth (nsubn nell n m) (i−(n−k))))
        by (simp add: 1 2 3 6 nfuse-nnth)
      have 43: i ≤ (m−k) ⟶ nnth (nfuse (nsubn nell k n) (nsubn nell n m)) i =
                 (if i≤ (n−k) then (nnth nell (k+i)) else (nnth nell (n+(i−(n−k)))))
        using assms 42 5 unfolding nsubn-def1
        by (auto simp add: ntaken-nnth)
           (metis enat-ord-simps(1) min.bounded-iff,
            metis enat-ord-simps(1) min.bounded-iff)
      have 44: i ≤ (m−k) ⟶ nnth (nfuse (nsubn nell k n) (nsubn nell n m)) i =
                 (nnth nell (k+i))
        by (metis 43 Nat.diff-diff-right Nat.le-diff-conv2 add.commute assms(1) enat-ord-simps(1)
           le-add-diff-inverse nat-le-linear)
      show ?thesis
        by (simp add: 41 44)
    qed
  qed
 have 8: (∀ i. i ≤ nlength(nsubn nell k m) ⟶ (nnth (nsubn nell k m) i) = (nnth nell (k+i)))
   using assms by (simp add: nsubn-def1 ntaken-nnth)
 show ?thesis
   by (simp add: 6 7 8 nellist-eq-nnth-eq)
qed

lemma nmap-nfuse:
 nmap f (nfuse nellx nelly) = nfuse (nmap f nellx) (nmap f nelly)
by (simp add: nfuse-def1 nmap-nappend-distrib)

lemma nzip-nfuse:
 assumes nlength σ1 = nlength l1
         nlength σ2 = nlength l2
         nfirst σ2 = nlast σ1
          nfinite σ1
          nfirst l2 = nlast l1
 shows  (nzip (nfuse σ1 σ2) (nfuse l1 l2)) = nfuse (nzip σ1 l1) (nzip σ2 l2)
using assms
apply transfer
by (simp add: epred-inject lfuse-def lzip-lappend)
```

1.3.7 nridx and nidx

lemma *nridx-nidx*:
nridx (<) *nell* = *nidx nell*
apply *transfer*
by (*simp add*: *ridx-lidx*)

lemma *nridx-expand*:
nridx R nell ⟷ (∀ *i*. (*Suc i*) ≤ *nlength nell* ⟶ *R* (*nnth nell i*) (*nnth nell* (*Suc i*)))
by *transfer*
(*auto simp add*: *min-def Suc-ile-eq*,
metis co.enat.exhaust-sel eSuc-enat ileI1 iless-Suc-eq llength-eq-0 ridx-def,
metis Extended-Nat.eSuc-mono co.enat.exhaust-sel eSuc-enat llength-eq-0 order-le-less ridx-def)

lemma *nidx-expand*:
nidx nell ⟷ (∀ *i*. (*Suc i*) ≤ *nlength nell* ⟶ *nnth nell i* < *nnth nell* (*Suc i*))
using *nridx-nidx nridx-expand* **by** *blast*

lemma *nridx-NCons*:
nridx R (*NCons x nell*) ⟷
(∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
using *nridx-expand*[*of R* (*NCons x nell*)]
by *auto*

lemma *nidx-NCons*:
nidx (*NCons x nell*) ⟷
(∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ (*nnth* (*NCons x nell*) *n*) < (*nnth* (*NCons x nell*) (*Suc n*)))
using *nridx-NCons*[*of* (<) *x nell*] *nridx-nidx* **by** *blast*

lemma *nridx-LCons-conv*:
(∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*))) ⟷
R x (*nfirst nell*) ∧
(∀ *n*. 0≤*n* ∧ (*Suc n*) ≤ (*nlength nell*) ⟶ *R* (*nnth nell n*) (*nnth nell* (*Suc n*)))
proof −
have *1*: (∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*))) ⟷
(∀ *n*. 0≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
by *blast*
have *2*: (∀ *n*. 0≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*))) ⟷
R x (*nfirst nell*) ∧
(∀ *n*. 1≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
by (*metis 1 One-nat-def add-diff-cancel-left′ eSuc-ile-mono enat-le-plus-same*(*1*) *gen-nlength-def le-SucE le-add1*
ndropn-Suc-NCons ndropn-nfirst ndropn-nfuse nfuse-leftneutral nlast-NNil nlength-NNil nlength-code nlength-eq-enat-nfiniteD nnth-0 one-eSuc one-enat-def plus-1-eq-Suc the-enat-0 zero-enat-def)
have *3*: *R x* (*nfirst nell*) ∧
(∀ *n*. 1≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*)

$(Suc\ n))) \longleftrightarrow$
$R\ x\ (nfirst\ nell) \wedge$
$(\forall n.\ 1{\leq}n \wedge (n) \leq\ (nlength\ nell)\ \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
by (*metis eSuc-enat eSuc-ile-mono*)
have *4*: $R\ x\ (nfirst\ nell) \wedge$
$(\forall n.\ 1{\leq}n \wedge (n) \leq\ (nlength\ nell)\ \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
$\longleftrightarrow$
$R\ x\ (nfirst\ nell) \wedge$
$(\forall n.\ 0{\leq}(n{-}1) \wedge (Suc(n{-}1)) \leq\ (nlength\ nell)\ \longrightarrow R\ (nnth\ \ nell\ (n{-}1))\ (nnth\ nell\ (Suc\ (n{-}1))))$
by (*metis add.commute add.right-neutral diff-add le-add1 nnth-Suc-NCons plus-1-eq-Suc*)
have *5*: $R\ x\ (nfirst\ nell) \wedge$
$(\forall n.\ 0{\leq}(n{-}1) \wedge (Suc(n{-}1)) \leq\ (nlength\ nell)\ \longrightarrow R\ (nnth\ \ nell\ (n{-}1))\ (nnth\ nell\ (Suc\ (n{-}1))))$
$\longleftrightarrow$
$R\ x\ (nfirst\ nell) \wedge$
$(\forall n.\ 0{\leq}n \wedge (Suc\ n) \leq\ (nlength\ nell)\ \longrightarrow R\ (nnth\ \ nell\ n)\ (nnth\ nell\ (Suc\ n)))$
by (*metis diff-Suc-1*)
show *?thesis*
using *1 2 3 4 5* **by** *presburger*
qed

lemma *nidx-LCons-conv*:
$(\forall n.\ (Suc\ n) \leq eSuc\ (nlength\ nell)\ \longrightarrow\ (nnth\ (NCons\ x\ nell)\ n) < (nnth\ (NCons\ x\ nell)\ (Suc\ n))) \longleftrightarrow$
$x < (nfirst\ nell) \wedge$
$(\forall n.\ 0{\leq}n \wedge (Suc\ n) \leq\ (nlength\ nell)\ \longrightarrow\ (nnth\ nell\ n) < (nnth\ nell\ (Suc\ n)))$
by (*metis nridx-LCons-conv*)

lemma *nridx-LCons-1* [*simp*]:
$nridx\ R\ (NCons\ x\ nell) \longleftrightarrow\ (\ R\ x\ (nfirst\ nell) \wedge nridx\ R\ nell\ )$
by (*metis nridx-LCons-conv nridx-NCons nridx-expand zero-le*)

lemma *nidx-LCons-1* [*simp*]:
$nidx\ \ (NCons\ x\ nell) \longleftrightarrow\ (\ \ x < (nfirst\ nell) \wedge nidx\ nell\ )$
by (*metis nridx-LCons-1 nridx-nidx*)

lemma *nridx-less*:
assumes $nridx\ R\ nell$
$Suc(n{+}k) \leq nlength\ nell$
$transp\ R$
shows $R\ (nnth\ nell\ n)\ (nnth\ nell\ (Suc(n{+}k)))$
using *assms*
proof (*induct k*)
case *0*
then show *?case*
by (*simp add*: *nridx-expand*)
next
case (*Suc k*)
then show *?case*
by (*metis add-Suc-right dual-order.trans eSuc-enat ile-eSuc nridx-expand transpE*)
qed

lemma *nidx-less*:
assumes *nidx nell*
Suc(*n*+*k*) ≤ *nlength nell*
shows *nnth nell n* < *nnth nell* (*Suc*(*n*+*k*))
using *assms*
by (*simp add*: *nridx-less nridx-nidx*)

lemma *nridx-less-eq*:
assumes *nridx R nell*
k ≤ *j*
j ≤ *nlength nell*
transp R
reflp R
shows *R* (*nnth nell k*) (*nnth nell j*)
proof (*cases k*=*j*)
case *True*
then show *?thesis* **using** *assms* **by** (*meson reflpE*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis* (*full-types*) *Suc-diff-Suc add.left-commute le-add-diff-inverse nridx-less order-neq-le-trans plus-1-eq-Suc*)
qed

lemma *nidx-less-eq*:
assumes *nidx nell*
k ≤ *j*
j ≤ *nlength nell*
shows (*nnth nell k*) ≤ (*nnth nell j*)
using *assms*
by (*metis Orderings.order-eq-iff antisym-conv2 less-iff-Suc-add nidx-less order.strict-implies-order*)

lemma *nridx-less-last*:
assumes *nridx R nell*
Suc i < *k*
nlength nell = (*enat k*)
transp R
shows *R* (*nnth nell i*) (*nnth nell* (*k*−*1*))
using *assms less-imp-Suc-add nridx-less* **by** *fastforce*

lemma *nidx-less-last*:
assumes *nidx nell*
Suc i < *k*
nlength nell = (*enat k*)
shows *nnth nell i* < *nnth nell* (*k*−*1*)
using *assms less-imp-Suc-add nidx-less* **by** *fastforce*

lemma *nidx-less-last-1*:
assumes *nidx nell*
i<*nlength nell*

nlength nell = (*enat k*)
shows *nnth nell i* < *nnth nell* (*k*)
using *assms*
by (*metis enat-ord-simps*(*2*) *less-imp-Suc-add linorder-le-cases nridx-less nridx-nidx transp-on-less*)

lemma *nridx-gr-first*:
assumes *nridx R nell*
0<*i*
i ≤ *nlength nell*
transp R
shows *R* (*nnth nell 0*) (*nnth nell i*)
using *assms nridx-less*[*of R nell 0 i*−*1*] **by** *simp*

lemma *nidx-gr-first*:
assumes *nidx nell*
0<*i*
i ≤ *nlength nell*
shows (*nnth nell 0*) < *nnth nell i*
using *assms nidx-less*[*of nell 0 i*−*1*]
by *simp*

lemma *nridx-ntake-a*:
assumes *nridx R nell*
n ≤ *nlength nell*
shows *nridx R* (*ntake n nell*)
using *assms*
by *transfer*
(*metis co.enat.exhaust-sel eSuc-ile-mono llength-eq-0 ridx-ltake-a*)

lemma *nidx-ntake-a*:
assumes *nidx nell*
n ≤ *nlength nell*
shows *nidx* (*ntake n nell*)
using *assms*
using *nridx-ntake-a nridx-nidx* **by** *blast*

lemma *nridx-nappend-nfinite*:
assumes *nfinite nell1*
shows *nridx R* (*nappend nell1 nell2*) ⟷
nridx R nell1 ∧ (*R* (*nlast nell1*) (*nfirst nell2*)) ∧ *nridx R nell2*
using *assms*
by *transfer*
(*simp add*: *ridx-lappend-lfinite*)

lemma *nidx-nappend-nfinite*:
assumes *nfinite nell1*
shows *nidx* (*nappend nell1 nell2*) ⟷
nidx nell1 ∧ ((*nlast nell1*) < (*nfirst nell2*)) ∧ *nidx nell2*

using *assms*
by (*metis nridx-nappend-nfinite nridx-nidx*)

lemma *nidx-nfuse*:
assumes *nfinite nell1*
nidx nell1
nnth nell1 0 = 0
nidx nell2
nnth nell2 0 = nlast nell1
shows *nidx* (*nfuse nell1 nell2*)
using *assms*
proof (*cases is-NNil nell2*)
case *True*
then show *?thesis* **unfolding** *nfuse-def1*
by (*simp add*: *assms*(*2*))
next
case *False*
then show *?thesis*
proof −
have *1*: *nlast nell1 = nfirst nell2*
by (*metis assms*(*5*) *ndropn-0 ndropn-nfirst*)
have *2*: *nidx* (*ntl nell2*)
by (*metis False assms*(*4*) *eSuc-enat ileI1 iless-Suc-eq nellist.collapse*(*2*) *nidx-expand*
nlength-NCons nnth-ntl)
have *3*: *nlast nell1 < nfirst*(*ntl nell2*)
by (*metis False assms*(*4*) *assms*(*5*) *eSuc-enat ileI1 iless-Suc-eq ndropn-0 ndropn-nfirst*
nellist.collapse(*2*) *nhd-conv-nnth nidx-expand nlength-NCons nnth-ntl zero-enat-def zero-le*)
have *4*: *nidx* (*nappend nell1* (*ntl nell2*))
by (*simp add*: *2 3 assms*(*1*) *assms*(*2*) *nidx-nappend-nfinite*)
show *?thesis* **using** *False* **unfolding** *nfuse-def1*
using *4* **by** *auto*
qed
qed

lemma *nridx-ndropn*:
assumes *nridx R nell*
n ≤ *nlength nell*
shows *nridx R* (*ndropn n nell*)
using *assms*
by *transfer*
(*metis co.enat.exhaust-sel iless-Suc-eq ldrop-enat llength-eq-0 min.orderE nless-le*
ridx-ldrop the-enat.simps)

lemma *nidx-ndropn*:
assumes *nidx nell*
n ≤ *nlength nell*
shows *nidx* (*ndropn n nell*)
using *assms*
using *nridx-ndropn nridx-nidx* **by** *blast*

lemma *nridx-ntake-all*:
assumes $\bigwedge n.\ n \leq$ *nlength nell* $\Longrightarrow$ *nridx R* (*ntake* (*enat n*) *nell*)
shows *nridx R nell*
using *assms*
by (*auto simp add*: *nridx-expand*)
(*metis Suc-ile-eq linorder-le-cases ntake-nnth ntake-nnth order-less-imp-le*)

lemma *nidx-ntake-all*:
assumes $\bigwedge n.\ n \leq$ *nlength nell* $\Longrightarrow$ *nidx* (*ntake* (*enat n*) *nell*)
shows *nidx nell*
using *assms*
using *nridx-nidx nridx-ntake-all* **by** *blast*

lemma *nridx-ntake*:
assumes *nridx R* (*ntake n nell*)
$n \leq$ *nlength nell*
$k \leq n$
shows *nridx R* (*ntake* (*enat k*) *nell*)
using *assms*
using *nridx-ntake-a* **by** *fastforce*

lemma *nidx-ntake*:
assumes *nidx* (*ntake n nell*)
$n \leq$ *nlength nell*
$k \leq n$
shows *nidx* (*ntake* (*enat k*) *nell*)
using *assms*
using *nridx-nidx nridx-ntake* **by** *blast*

lemma *nidx-imp-ndistinct*:
assumes *nidx nell*
shows *ndistinct nell*
using *assms*
apply *transfer*
using *lidx-imp-ldistinct* **by** *auto*

lemma *ndistinct-Ex1*:
assumes *ndistinct nell*
$x \in$ *nset nell*
shows $\exists!i.\ i \leq$ *nlength nell* $\wedge$ (*nnth nell i*) $= x$
using *assms*
by *transfer*
(*auto*,
metis co.enat.exhaust-sel iless-Suc-eq ldistinct-Ex1 llength-eq-0 min.orderE the-enat.simps,
metis co.enat.exhaust-sel iless-Suc-eq ldistinct-conv-lnth llength-eq-0)

lemma *nidx-nset-eq*:
assumes *nidx nellx*

nidx nelly
nset nellx = nset nelly
shows *nellx = nelly*
using *assms*
by *transfer*
(*simp add*: *bi-unique-cr-nellist-help lidx-lset-eq nidx.rep-eq*)

lemma *nridx-nfuse-nfirst-nlast*:
assumes *nridx R nell1*
(*nnth nell1 0*) = (*0*::*nat*)
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell
nlast nell1 = cp
shows *nlast nell1 = nfirst*(*nmap* (*λx. x+cp*) *nell2*)
using *assms*
by (*metis add.commute add.right-neutral ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le*)

lemma *nidx-nfuse-nfirst-nlast*:
assumes *nidx nell1*
(*nnth nell1 0*) = (*0*::*nat*)
nidx nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell
nlast nell1 = cp
shows *nlast nell1 = nfirst*(*nmap* (*λx. x+cp*) *nell2*)
using *assms*
by (*metis add.commute add.right-neutral ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le*)

lemma *nridx-nfuse-nnth-cp*:
assumes *nridx R nell1*
(*nnth nell1 0*) = *0*
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = cp
nlast nell2 = the-enat((*nlength nell*)) − *cp*
i ≤ (*nlength nell2*)
cp ≤ *nlength nell*
shows *nnth* (*nfuse nell1* (*nmap* (*λx. x+cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) = *cp* + (*nnth nell2 i*)
proof −
have *1*: *nlast nell1 = nfirst* (*nmap* (*λx. x+cp*) *nell2*)
by (*metis add-0 assms*(*4*) *assms*(*8*) *ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le*)
have *2*: *nnth* (*nfuse nell1* (*nmap* (*λx. x+cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) =
nnth (*nmap* (*λx. x+cp*) *nell2*) *i*
by (*simp add*: *1 assms nfuse-nnth-a*)

have *3*: *nnth* (*nmap* (λ*x*. *x*+*cp*) *nell2*) *i* = *cp* + (*nnth nell2 i*)
by (*simp add*: *assms*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *nidx-nfuse-nnth-cp*:
assumes *nidx nell1*
(*nnth nell1 0*) = *0*
nidx nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*
nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ (*nlength nell2*)
cp ≤ *nlength nell*
shows *nnth* (*nfuse nell1* (*nmap* (λ*x*. *x*+*cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) = *cp* + (*nnth nell2 i*)
using *assms nridx-nfuse-nnth-cp nridx-nidx* **by** *blast*

lemma *nridx-nfuse-nnth-cp-a*:
assumes *nridx R nell1*
(*nnth nell1 0*) = (*0*::*nat*)
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*
nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ ((*nlength nell1*)) + (*nlength nell2*)
the-enat((*nlength nell1*)) ≤ *i*
cp ≤ *nlength nell*
shows *nnth* (*nfuse nell1* (*nmap* (λ*x*. *x*+*cp*) *nell2*)) (*i*) = *cp* + (*nnth nell2* (*i* −*the-enat*((*nlength nell1*))))

proof −
have *1*: *i* = (*the-enat* ((*nlength nell1*)) + (*i* − *the-enat* ((*nlength nell1*))))
by (*simp add*: *assms*)
have *2*: *enat* ((*i*::*nat*) − *the-enat* ((*nlength nell1*))) ≤ *nlength nell2*
using *assms*
by (*metis enat-add-sub-same enat-minus-mono1 enat-ord-simps*(*1*)
idiff-enat-enat infinity-ileE nfinite-conv-nlength-enat the-enat.simps)
show *?thesis* **using** *1 2 assms nridx-nfuse-nnth-cp*[*of R nell1 nell2 nell cp* (*i* − *the-enat* ((*nlength nell1*)))]
by *presburger*
qed

lemma *nidx-nfuse-nnth-cp-a*:
assumes *nidx nell1*

$(nnth\ nell1\ 0\ ) = (0::nat)$
$nidx\ \ nell2$
$(nnth\ nell2\ 0\ ) = 0$
$nfinite\ nell1$
$nfinite\ nell2$
$nfinite\ nell$
$nlast\ nell1 = cp$
$nlast\ nell2 = the\text{-}enat((nlength\ nell)) - cp$
$i \leq ((nlength\ nell1)) + (nlength\ nell2)$
$the\text{-}enat((nlength\ nell1)) \leq i$
$cp \leq nlength\ nell$

shows $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+cp)\ nell2))\ (\ i) = cp + (nnth\ nell2\ (i - the\text{-}enat((nlength\ nell1))))$

using *assms nridx-nidx nridx-nfuse-nnth-cp-a* **by** *blast*

lemma *nridx-nfuse-nnth-cp-nlast*:
assumes $nridx\ R\ nell1$
$(nnth\ nell1\ 0\ ) = 0$
$nridx\ R\ nell2$
$(nnth\ nell2\ 0\ ) = 0$
$nfinite\ nell1$
$nfinite\ nell2$
$nfinite\ nell$
$nlast\ nell1 = cp$
$nlast\ nell2 = the\text{-}enat(\ (nlength\ nell)) - cp$
$i \leq (nlength\ nell2)$
$cp \leq nlength\ nell$
shows $nlast\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+cp)\ nell2)) = (the\text{-}enat\ ((nlength\ nell)))$
proof −
have *1*: $nlast\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+cp)\ nell2)) = nlast\ (nmap\ (\lambda x.\ x+cp)\ nell2)$
using *assms*
by (*metis add-0 ndropn-0 ndropn-nfirst nlast-nfuse nnth-nmap zero-enat-def zero-le*)
have *2*: $nlast\ (nmap\ (\lambda x.\ x+cp)\ nell2) = cp + (nlast\ nell2)$
by (*simp add*: *assms*(*6*))
have *3*: $cp + (nlast\ nell2) = (the\text{-}enat\ ((nlength\ nell)))$
using *assms*
by (*metis add.commute diff-add enat-ord-simps*(*1*) *nfinite-conv-nlength-enat the-enat.simps*)
show *?thesis*
by (*simp add*: *1 3 add.commute assms*(*6*))
qed

lemma *nidx-nfuse-nnth-cp-nlast*:
assumes $nidx\ \ nell1$
$(nnth\ nell1\ 0\ ) = 0$
$nidx\ \ nell2$
$(nnth\ nell2\ 0\ ) = 0$
$nfinite\ nell1$
$nfinite\ nell2$
$nfinite\ nell$
$nlast\ nell1 = cp$

nlast nell2 = the-enat((nlength nell)) − cp
i ≤ (nlength nell2)
cp ≤ nlength nell
shows *nlast (nfuse nell1 (nmap (λx. x+cp) nell2)) = (the-enat ((nlength nell)))*
using *assms nridx-nidx nridx-nfuse-nnth-cp-nlast* **by** *blast*

lemma *nridx-nfuse-nnth-cp-infinite*:
assumes *nridx R nell1*
(nnth nell1 0) = (0::nat)
nridx R nell2
(nnth nell2 0) = 0
nfinite nell1
¬nfinite nell2
¬nfinite nell
nlast nell1 = cp
shows *nnth (nfuse nell1 (nmap (λx. x+cp) nell2)) (the-enat((nlength nell1)) + i) = cp + (nnth nell2 i)*
proof −
have *1*: *nlast nell1 = nfirst(nmap (λx. x+cp) nell2)*
by (*metis assms(1) assms(2) assms(3) assms(4) assms(5) assms(8) nridx-nfuse-nfirst-nlast*)
have *2*: *nnth (nfuse nell1 (nmap (λx. x+cp) nell2)) (the-enat((nlength nell1)) + 0) = nlast nell1*
using *assms* **by** (*metis 1 add.right-neutral ntaken-nfuse ntaken-nlast*)
have *3*: *nfirst(nmap (λx. x+cp) nell2) = cp + (nnth nell2 0)*
using *1 assms* **by** *auto*
have *8*: *i≤ (nlength nell2)*
by (*metis assms(6) enat-ile nfinite-conv-nlength-enat wlog-linorder-le*)
have *10*: *(nlength nell1) ≤ enat (the-enat ((nlength nell1)) + (i::nat))*
using *assms(5) nfinite-nlength-enat* **by** *fastforce*
have *11*: *enat (the-enat ((nlength nell1)) + i) ≤ nlength (nfuse nell1 (nmap (λx::nat. x + cp) nell2))*
by (*metis 10 assms(6) enat-less-enat-plusI2 enat-ord-code(4) enat-the-enat leD nfuse-nlength nlength-eq-enat-nfiniteD nlength-nmap order-less-imp-le*)
have *12*: *(the-enat ((nlength nell1)) + i − the-enat ((nlength nell1))) = i*
by *auto*
have *4*: *nnth (nfuse nell1 (nmap (λx. x+cp) nell2)) (the-enat((nlength nell1)) + i) =*
(nnth (nmap (λx. x+cp) nell2) i)
by (*simp add*: *1 8 assms nfuse-nnth-a*)
have *5*: *(nnth (nmap (λx. x+cp) nell2) i) = cp + (nnth nell2 i)*
by (*simp add*: *8*)
show *?thesis*
using *4 5* **by** *presburger*
qed

lemma *nidx-nfuse-nnth-cp-infinite*:
assumes *nidx nell1*
(nnth nell1 0) = 0
nidx nell2
(nnth nell2 0) = 0
nfinite nell1
¬nfinite nell2
¬nfinite nell
nlast nell1 = cp

shows *nnth (nfuse nell1 (nmap (λx. x+cp) nell2)) (the-enat((nlength nell1)) + i) = cp + (nnth nell2 i)*
using *assms nridx-nidx nridx-nfuse-nnth-cp-infinite* **by** *blast*

lemma *nidx-nfuse-nidx*:
assumes *nidx nell1*
nnth nell1 0 = 0
nidx nell2
nnth nell2 0 = 0
nfinite nell1
nlast nell1 = cp
nfinite nell2
nfinite nell
nlast nell2 = the-enat((nlength nell)) −cp
cp ≤ nlength nell
shows *nidx (nfuse nell1 (nmap (λx. x+ cp) nell2)) ∧ (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) 0) = 0*
proof −
have *1*: *nlast nell1 = nfirst(nmap (λx. x+ cp) nell2)*
using *assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) nidx-nfuse-nfirst-nlast* **by** *blast*
have *2*: *nfirst (nfuse nell1 (nmap (λx. x+ cp) nell2)) = nfirst nell1*
by (*simp add*: *1 nfirst-nfuse*)
have *4*: *⋀j. j≤nlength nell1 ⟹ nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j= nnth nell1 j*
using *assms* **by** (*simp add*: *1 add-increasing2 nfuse-nlength nfuse-nnth*)
have *40*: *∃ k1. nlength nell1 = (enat k1)*
by (*simp add*: *assms(5) nfinite-nlength-enat*)
obtain *k1* **where** *41*: *nlength nell1 = (enat k1)*
using *40* **by** *blast*
have *5*: *⋀j. (nlength nell1) ≤j ∧ j ≤(nlength nell1) + nlength nell2 ⟹*
nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j =
cp + (nnth nell2 (j− the-enat((nlength nell1))))
using *assms nidx-nfuse-nnth-cp-a*[*of nell1 nell2 nell cp*]
by (*metis 41 enat-ord-simps(1) the-enat.simps*)
have *45*: *⋀j. k1 ≤j ∧ j ≤ (enat (k1)) + nlength nell2 ⟹*
nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j =
cp + (nnth nell2 (j− (k1)))
by (*simp add*: *41 5 order-less-imp-le*)
have *50*: *nlength(nfuse nell1 (nmap (λx. x+ cp) nell2)) = (nlength nell1) + nlength nell2*
by (*simp add*: *nfuse-nlength*)
have *51*: *⋀j. enat (Suc j) ≤ nlength nell1 ⟹*
(nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) <
(nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) (Suc j))
using *4 Suc-ile-eq assms(1) nidx-expand* **by** *auto*
have *52*: *⋀j. k1 ≤j ∧ (Suc j) ≤ enat (k1) + nlength nell2 ⟹*
cp + (nnth nell2 (j− (k1))) <
cp + (nnth nell2 ((Suc j)− (k1)))
using *assms(3)* **unfolding** *nidx-def*
by (*metis Nat.add-diff-assoc add-strict-left-mono assms(3) enat.simps(3) enat-add-sub-same enat-minus-mono1 idiff-enat-enat nidx-expand plus-1-eq-Suc*)
have *6*: *⋀j. enat (Suc j) ≤ nlength(nfuse nell1 (nmap (λx. x+ cp) nell2)) ⟹*
(nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) <

```
            (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) (Suc j))
    proof −
      fix j
      assume a: enat (Suc j) ≤ nlength(nfuse nell1 (nmap (λx. x+ cp) nell2))
       show (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) < (nnth (nfuse nell1 (nmap (λx. x+ cp)
nell2)) (Suc j))
      proof −
       have 61: enat (Suc j) ≤ nlength nell1 ⟹
          (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) < (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2))
(Suc j))
         using 51 by blast
       have 62: k1 ≤j ∧ (Suc j) ≤ nlength(nfuse nell1 (nmap (λx. x+ cp) nell2)) ⟹
          (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) < (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2))
(Suc j))
         using 41 5 50 52 Suc-ile-eq by force
       show ?thesis
       using 41 61 62 a by fastforce
      qed
   qed
 show ?thesis
 unfolding nidx-expand
   using 4 6 assms zero-enat-def by force
qed

lemma nidx-nfuse-nidx-infinite:
 assumes nidx  nell1
        nnth nell1 0 = 0
        nidx  nell2
        nnth nell2 0 = 0
        nfinite nell1
        nlast nell1  = cp
        ¬nfinite nell2
        ¬nfinite nell
 shows   nidx (nfuse nell1 (nmap (λx. x+ cp) nell2)) ∧ (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) 0)
= 0
proof −
 have 1: nlast nell1 = nfirst(nmap (λx. x+ cp) nell2)
   by (metis add-0 assms(4) assms(6) assms(7) enat-le-plus-same(1) enat-le-plus-same(2) enat-the-enat
      infinity-ileE ndropn-0 ndropn-nfirst nfinite-conv-nlength-enat nnth-nmap)
 have 2: nfirst (nfuse nell1 (nmap (λx. x+ cp) nell2)) = nfirst nell1
   by (simp add: 1 nfirst-nfuse)
 have 4: ⋀j. j≤nlength nell1 ⟹ nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j= nnth nell1 j
   by (simp add: 1 add-increasing2 nfuse-nlength nfuse-nnth)
 have 40: ∃ k1. nlength nell1 = (enat k1)
   by (simp add: assms(5) nfinite-nlength-enat)
 obtain k1 where 41: nlength nell1 = (enat k1)
   using 40 by blast
 have 5: ⋀j.  (nlength nell1) ≤j ∧ j ≤ (nlength nell1) + nlength nell2 ⟹
        nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j =
        cp + (nnth nell2 (j− the-enat((nlength nell1))))
```

using *assms nidx-nfuse-nnth-cp-infinite*[*of nell1 nell2 nell cp*]
by (*metis 41 enat-ord-simps*(1) *le-add-diff-inverse the-enat.simps*)
have *45*: $\bigwedge j.\ \ k1 \leq j \wedge j \leq (enat\ (k1)) + nlength\ nell2 \Longrightarrow$
nnth (*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) *j* =
cp + (*nnth nell2* (*j*− (*k1*)))
using *41 5 enat-ord-simps*(1) *the-enat.simps* **by** *presburger*
have *50*: *nlength*(*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) = (*nlength nell1*) + *nlength nell2*
by (*simp add*: *nfuse-nlength*)
have *51*: $\bigwedge j.$ *enat* (*Suc j*) ≤ *nlength nell1* ⟹
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) *j*) <
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) (*Suc j*))
using *4 Suc-ile-eq assms*(1) *nidx-expand* **by** *auto*
have *52*: $\bigwedge j.\ \ k1 \leq j \wedge (Suc\ j) \leq enat\ (k1) + nlength\ nell2 \Longrightarrow$
cp + (*nnth nell2* (*j*− (*k1*))) <
cp + (*nnth nell2* ((*Suc j*)− (*k1*)))
by (*metis Nat.add-diff-assoc add-strict-left-mono assms*(3) *enat.simps*(3) *enat-add-sub-same*
enat-minus-mono1 idiff-enat-enat nidx-expand plus-1-eq-Suc)
have *53*: $\bigwedge j.\ \ k1 \leq j \wedge (Suc\ j) \leq enat\ (k1) + nlength\ nell2 \Longrightarrow$
nnth (*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) *j* <
nnth (*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) (*Suc j*)
by (*metis 45 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le*)
have *6*: $\bigwedge j.$ *enat* (*Suc j*) ≤ *nlength*(*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) ⟹
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) *j*) <
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+\ cp$) *nell2*)) (*Suc j*))
by (*metis 41 50 51 53 enat-ord-simps*(1) *le-SucE linorder-le-cases*)
show *?thesis* **unfolding** *nidx-expand*
using *4 6 assms zero-enat-def* **by** *fastforce*
qed

lemma *nsubn-nfuse-nidx*:
assumes *nidx nl*
nfinite nl
nfinite nell
nlast nl = (*nlength nell*)
(*Suc i*) ≤ (*nlength nl*)
shows *nfuse* (*nsubn nell* (*nnth nl i*) (*nnth nl* (*Suc i*))) (*nsubn nell* (*nnth nl* (*Suc i*)) (*nlast nl*)) =
(*nsubn nell* (*nnth nl i*) (*nlast nl*))
proof −
have *1*: (*nnth nl i*) ≤ (*nnth nl* (*Suc i*))
by (*simp add*: *assms*(1) *assms*(5) *nidx-less-eq*)
have *2*: *nlast nl* = (*nnth nl* (*the-enat*((*nlength nl*))))
by (*simp add*: *assms*(2) *nnth-nlast*)
have *3*: *1* ≤ *nlength nl*
by (*metis assms*(5) *dual-order.trans enat-0-iff*(1) *iless-Suc-eq le-zero-eq linorder-le-cases*
nat.simps(3) *one-eSuc order-neq-le-trans*)
have *4*: (*nnth nl* (*Suc i*)) ≤ (*nlast nl*)
by (*metis 2 assms*(1) *assms*(2) *assms*(5) *dual-order.eq-iff enat-ord-simps*(1)
nfinite-conv-nlength-enat nidx-less-eq the-enat.simps)
have *5*: *enat* (*nnth nl* (*Suc i*)) ≤ *nlength nell*

by (*metis 4 assms(4) enat-ord-simps(1)*)
have *6*: *nlast* (*nsubn nell* (*nnth nl i*) (*nnth nl* (*Suc i*))) = (*nnth nell* (*nnth nl* (*Suc i*)))
by (*simp add*: *1 nsubn-def1 ntaken-nlast*)
have *7*: *nfirst* (*nsubn nell* (*nnth nl* (*Suc i*)) (*nlast nl*)) = (*nnth nell* (*nnth nl* (*Suc i*)))
by (*simp add*: *nsubn-def1 ntaken-ndropn-nfirst*)
show *?thesis*
by (*simp add*: *1 5 assms(4) nsubn-nfuse*)
qed

lemma *nidx-nfuse-split*:
assumes *nlast nell1* = *nfirst nell2*
shows *nridx R* (*nfuse nell1 nell2*) ⟷
(*if nfinite nell1 then nridx R nell1* ∧ *nridx R nell2 else nridx R nell1*)
proof (*cases nfinite nell1*)
case *True*
then show *?thesis*
by (*metis assms nappend-nbutlast-nlast-id nbutlast-nfinite nellist.collapse*(*2*) *nellist.disc*(*1*)
nfuse-def1 nfuse-leftneutral nhd-nfuse nlast-NNil nridx-LCons-1 nridx-nappend-nfinite)
next
case *False*
then show *?thesis* **by** (*simp add*: *assms nfuse-nappend*)
qed

lemma *nidx-all-le-nlast*:
assumes *nidx nell*
nfinite nell
j≤ *nlength nell*
shows *nnth nell j* ≤ *nlast nell*
using *assms*
by (*metis nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast order.order-iff-strict the-enat.simps*)

lemma *nidx-shiftm* :
assumes *nidx nell*
nnth nell 0 = *k*
shows *nidx* (*nmap* (λ*x* .*x*− *k*) *nell*) ∧ *nnth* (*nmap* (λ*x* .*x*− *k*) *nell*) *0* = *0* ∧ *k* ≤ (*nnth nell 0*)
using *assms zero-enat-def*
by (*auto simp add*: *nidx-expand*)
(*metis Suc-ile-eq add-diff-inverse-nat add-gr-0 assms*(*1*) *diff-less-mono nidx-gr-first*
nnth-nmap order-less-imp-le zero-less-Suc zero-less-diff)

lemma *nidx-nsubn*:
assumes *k*≤*n*
n ≤ *nlength nell*
nidx nell
nnth nell 0 = *0*
shows *nidx* (*nsubn nell k n*) ∧ *nnth* (*nsubn nell k n*) *0* = (*nnth nell k*)
using *assms* **unfolding** *nidx-expand nsubn-def1*
by (*auto simp add*: *ntaken-nnth*)
(*simp add*: *Nat.le-diff-conv2 add.commute order-subst2*)

lemma *nidx-ntaken-niterates-Suc*:
nidx (*ntaken n* (*niterates Suc 0*))
proof −
have *1*: *nidx* (*ntaken n* (*niterates Suc 0*)) =
(∀ *i*::*nat*.
enat (*Suc i*) ≤ *nlength* (*ntaken n* (*niterates Suc* (*0*::*nat*))) ⟶
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) *i* <
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) (*Suc i*))
unfolding *nidx-expand* **by** *auto*
have *2*: (∀ *i*::*nat*.
enat (*Suc i*) ≤ *nlength* (*ntaken n* (*niterates Suc* (*0*::*nat*))) ⟶
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) *i* <
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) (*Suc i*))
proof
fix *i*
show *enat* (*Suc i*) ≤ *nlength* (*ntaken n* (*niterates Suc* (*0*::*nat*))) ⟶
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) *i* <
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) (*Suc i*)
proof (*induct i arbitrary*: *n*)
case *0*
then show *?case*
by (*simp add*: *ntaken-nnth*)
next
case (*Suc i*)
then show *?case* **by** (*simp add*: *ntaken-nnth*)
qed
qed
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *forall-nappend-single*:
assumes *nfinite l*
shows (∀ *i*< *nlength* (*nappend l* (*NNil x*)). *P i* (*Suc i*)) ⟷
(*P* (*the-enat*(*nlength l*)) (*Suc*(*the-enat*(*nlength l*))) ∧ (∀ *i*< *nlength l*. *P i* (*Suc i*)))
proof (*cases nlength l* = *0*)
case *True*
then show *?thesis* **using** *assms*
by (*metis enat-0-iff*(*1*) *gr-implies-not-zero iless-eSuc0 nappend-NNil*
ndropn-0 ndropn-nlast nlength-NCons nlength-NNil the-enat-0)
next
case *False*
then show *?thesis* **using** *assms*
by (*simp add*: *enat-the-enat nfinite-conv-nlength-enat*)
(*metis dual-order.order-iff-strict iless-Suc-eq plus-1-eSuc*(*2*) *the-enat.simps*)
qed

lemma *forall-ncons-split*:
assumes *nfinite l*
shows (∀ *i*. *enat i* < *nlength* (*NCons 0 l*) ⟶
P (*nnth* (*NCons 0 l*) *i*) (*nnth* (*NCons 0 l*) (*Suc i*))) ⟷
P 0 (*nnth l 0*) ∧
(∀ *i*. *enat i* < *nlength l* ⟶
P (*nnth l i*) (*nnth l* (*Suc i*)))
using *assms*
by (*auto simp add*: *zero-enat-def*)
(*metis nnth-0 zero-enat-def zero-le*,
metis eSuc-enat ileI1 nnth-Suc-NCons,
metis Suc-ile-eq Suc-pred less-le-not-le nle-le nnth-0 nnth-Suc-NCons zero-le)

lemma *forall-ncons-split-alt*:
assumes ¬*nfinite l*
shows (∀ *i*. *enat i* < *nlength* (*NCons 0 l*) ⟶
P (*nnth* (*NCons 0 l*) *i*) (*nnth* (*NCons 0 l*) (*Suc i*))) ⟷
P 0 (*nnth l 0*) ∧
(∀ *i*. *enat i* < *nlength l* ⟶
P (*nnth l i*) (*nnth l* (*Suc i*)))
using *assms*
by *auto*
(*metis nnth-0 zero-enat-def zero-le*,
metis eSuc-enat ileI1 nnth-Suc-NCons,
metis Suc-ile-eq Suc-pred le-zero-eq less-le-not-le nle-le nnth-0 nnth-Suc-NCons)

lemma *nidx-Ncons-shift*:
assumes *nfinite l*
nidx l
shows *nidx* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
proof −
have *1*: *nidx* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟷
0 < *nfirst* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*) ∧
nidx (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
by *auto*
have *2*: *0* < *nfirst* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
by (*metis Zero-not-Suc add-Suc-right bot-nat-0.not-eq-extremum enat-defs*(*1*) *ndropn-0*
ndropn-nfirst nnth-nmap zero-le)
have *3*: *nidx* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
using *nidx-expand Suc-ile-eq assms*(*2*) **by** *force*
show *?thesis*
using *2 3* **by** *auto*
qed

lemma *nidx-Ncons-shift-alt*:
assumes ¬*nfinite l*
nidx l
shows *nidx* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
proof −

have *1*: *nidx* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟷
0 < *nfirst* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*) ∧
nidx (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
by *auto*
have *2*: *0* < *nfirst* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
by (*metis Zero-not-Suc add-Suc-right bot-nat-0.not-eq-extremum enat-defs*(*1*) *ndropn-0 ndropn-nfirst nnth-nmap zero-le*)
have *3*: *nidx* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
using *nidx-expand Suc-ile-eq assms*(*2*) **by** *force*
show *?thesis*
using *2 3* **by** *auto*
qed

lemma *infinite-nidx-imp-infinite-interval*:
assumes ¬ *nfinite l*
nidx l
(*nnth l 0*) = *0*
∀ *i*. (*nnth l i*) ≤ *nlength s*
shows ¬ *nfinite s*
proof
assume *nfinite s*
thus *False*
using *assms*
proof (*induct s rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis dual-order.antisym enat-ord-simps*(*1*) *linorder-linear nfinite-ntaken nidx-gr-first nlength-NNil not-gr-zero ntaken-all zero-le zero-less-Suc*)
next
case (*NCons x nell*)
then show *?case*
proof −
have *1*: ⋀*j*. (*nnth l j*) < (*nnth l* (*Suc j*))
by (*metis assms*(*1*) *assms*(*2*) *linorder-le-cases nfinite-ntaken nidx-expand ntaken-all*)
have *2*: ∀ *i*. *enat* (*nnth l i*) ≤ *nlength* (*NCons x nell*) ⟹ *False*
by (*metis 1 NCons.hyps*(*2*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *dual-order.strict-iff-order enat-ord-simps*(*1*) *iless-Suc-eq linorder-not-le nlength-NCons*)
show *?thesis*
using *2 NCons.prems*(*4*) **by** *auto*
qed
qed
qed

1.3.8 nlastnfirst

lemma *nlastnfirst-def2*:
nlastnfirst = *llastlfirst* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
by (*simp add*: *map-fun-def nlastnfirst-def*)

lemma *nlastnfirst-NNil*[*simp*]:

nlastnfirst (*NNil x*)
apply *transfer*
by *simp*

lemma *nlastnfirst-LCons*[*simp*]:
shows *nlastnfirst* (*NCons nell nells*) ⟷
nlast nell = *nfirst* (*nfirst nells*) ∧ *nlastnfirst nells*
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlastnfirst* (*NCons nell nells*) =
(*llastlfirst* ∘ *?n2l*) (*NCons nell nells*)
by (*simp add*: *nlastnfirst-def2*)
have *2*: (*llastlfirst* ∘ *?n2l*) (*NCons nell nells*) =
(*llast* (*llist-of-nellist nell*) = *lfirst* (*lfirst* (*?n2l nells*)) ∧ *llastlfirst* (*?n2l nells*))
by *simp*
have *3*: *llastlfirst* (*?n2l nells*) = *nlastnfirst nells*
by (*simp add*: *nlastnfirst.rep-eq*)
have *4*: *llast* (*llist-of-nellist nell*) = *nlast nell*
by (*metis llast-linfinite nfinite-def nlast-llast nlast-not-nfinite*)
have *5*: *lfirst* (*lfirst* (*?n2l nells*)) = *nfirst* (*nfirst nells*)
by (*simp add*: *lfirst-def llist-of-nellist.simps*(*2*) *nfirst-def*)
show *?thesis*
using *1 2 3 4 5* **by** *presburger*
qed

lemma *nlastnfirst-def1*:
shows *nlastnfirst nells* =
(∀ *i*. (*Suc i*)≤ *nlength nells* ⟶ *nlast*(*nnth nells i*) = *nfirst*(*nnth nells* (*Suc i*)))
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlastnfirst nells* = (*llastlfirst* ∘ *?n2l*) *nells*
by (*simp add*: *nlastnfirst-def2*)
have *2*: (*llastlfirst* ∘ *?n2l*) *nells* = *llastlfirst* (*?n2l nells*)
by *simp*
have *3*: *llastlfirst* (*?n2l nells*) ⟷
(∀ *i*. (*Suc i*) < *llength* (*?n2l nells*) ⟶
llast (*lnth* (*?n2l nells*) *i*) = *lfirst* (*lnth* (*?n2l nells*) (*Suc i*)))
using *llastlfirst-def* **by** *blast*
have *4*: *epred* (*llength* (*?n2l nells*)) = *nlength nells*
by *simp*
have *5*: ⋀*xs j*. *j*≤ *nlength xs* ⟶ *nnth xs j* = *lnth* (*llist-of-nellist xs*) *j*
unfolding *nnth-def* **by** *auto*
have *6*: ⋀*lx j* . *j*<*llength lx* ∧ ¬ *lnull lx*⟶ *lnth lx j* = *nnth* (*nellist-of-llist lx*) *j*
by (*metis 5 co.enat.exhaust-sel iless-Suc-eq llength-eq-0 nellist-of-llist-inverse nlength.abs-eq*)
have *7*: ⋀*xs*. *nlast xs* = *llast* (*llist-of-nellist xs*)
by (*metis llast-linfinite nfinite-def nlast-llast nlast-not-nfinite*)
have *8*: ⋀*xs*. *nfirst xs* = *lfirst* (*llist-of-nellist xs*)
by (*simp add*: *lfirst-def llist-of-nellist.simps*(*2*) *nfirst-def*)
have *9*: ⋀*lx*. ¬ *lnull lx* ⟹ *llast lx* = *nlast* (*nellist-of-llist lx*)

by (*simp add*: *7*)
have *10*: $\bigwedge lx.\ \neg$ *lnull lx* $\Longrightarrow$ *lfirst lx* = *nfirst* (*nellist-of-llist lx*)
by (*simp add*: *8*)
have *11*: $\bigwedge$ *j*. *j* < *llength* (*?n2l nells*) $\longrightarrow$
llast (*lnth* (*?n2l nells*) *j*) =
nlast (*nellist-of-llist* (*lnth* (*?n2l nells*) *j*))
by (*simp add*: *9*)
have *12*: $\bigwedge$ *j*. (*enat j*) < *llength* (*?n2l nells*) $\longrightarrow$
nlast (*nellist-of-llist* (*lnth* (*?n2l nells*) *j*)) =
nlast (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) *j*)))
by *simp*
have *13*: $\bigwedge$ *j*. (*enat j*) < *llength* (*?n2l nells*) $\longrightarrow$
nlast (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) *j*))) =
nlast (*lnth* (*llist-of-nellist nells*) *j*)
by *auto*
have *14*: $\bigwedge$ *j*. (*enat j*) < *llength* (*?n2l nells*) $\longrightarrow$
nlast (*lnth* (*llist-of-nellist nells*) *j*) = *nlast*(*nnth nells j*)
by (*simp add*: *6*)
have *15*: $\bigwedge$ *j*. *j* < *llength* (*?n2l nells*) $\longrightarrow$
llast (*lnth* (*?n2l nells*) *j*) =
nlast(*nnth nells j*)
using *11 12 13 14* **by** *presburger*
have *16*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) $\longrightarrow$
lfirst (*lnth* (*?n2l nells*) (*Suc j*)) =
nfirst (*nellist-of-llist* (*lnth* (*?n2l nells*) (*Suc j*)))
by (*simp add*: *10*)
have *17*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) $\longrightarrow$
nfirst (*nellist-of-llist* (*lnth* (*?n2l nells*) (*Suc j*))) =
nfirst (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) (*Suc j*))))
by *simp*
have *18*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) $\longrightarrow$
nfirst (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) (*Suc j*)))) =
nfirst (*lnth* (*llist-of-nellist nells*) (*Suc j*))
by *auto*
have *19*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) $\longrightarrow$
nfirst (*lnth* (*llist-of-nellist nells*) (*Suc j*)) = *nfirst* (*nnth nells* (*Suc j*))
by (*simp add*: *6*)
have *20*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) $\longrightarrow$
lfirst (*lnth* (*?n2l nells*) (*Suc j*)) = *nfirst* (*nnth nells* (*Suc j*))
using *16 17 18 19* **by** *presburger*
have *21*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) $\longleftrightarrow$
(*Suc j*) $\leq$ *nlength nells*
by (*metis 4 co.enat.exhaust-sel dual-order.strict-iff-order enat-0-iff*(*1*) *epred-0 iless-Suc-eq*
nat.simps(*3*))
have *22*: ($\forall$ *i*. (*Suc i*) < *llength* (*?n2l nells*) $\longrightarrow$
llast (*lnth* (*?n2l nells*) *i*) = *lfirst* (*lnth* (*?n2l nells*) (*Suc i*)))
$\longleftrightarrow$
($\forall$ *i*. (*Suc i*)$\leq$ *nlength nells* $\longrightarrow$ *nlast*(*nnth nells i*) = *nfirst*(*nnth nells* (*Suc i*)))
by (*metis 15 20 21 Suc-ile-eq order-less-imp-le*)
have *23*: *llastlfirst* (*?n2l nells*) $\longleftrightarrow$

$(\forall\, i.\ (Suc\ i) \leq nlength\ nells \longrightarrow nlast(nnth\ nells\ i) = nfirst(nnth\ nells\ (Suc\ i)))$
using *22 3* **by** *presburger*
show *?thesis* **using** *1 23* **by** *auto*
qed

lemma *nlastnfirst-nridx*:
$nlastnfirst\ nells = nridx\ (\lambda\ a\ b.\ nlast\ a = nfirst\ b)\ nells$
by (*simp add*: *nlastnfirst-def1 nridx-expand*)

lemma *nlastnfirst-nappend-nfinite*:
assumes *nfinite nellxs*
shows $nlastnfirst\ (nappend\ nellxs\ nellys) \longleftrightarrow$
$nlastnfirst\ nellxs \wedge nlastnfirst\ nellys \wedge nlast(nlast\ nellxs) = nfirst(nfirst\ nellys)$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *nlastnfirst-nridx nridx-nappend-nfinite*)

1.3.9 nfusecat

lemma *nfusecat-def2*:
$nfusecat = nellist\text{-}of\text{-}llist \circ lfusecat \circ (lmap\ llist\text{-}of\text{-}nellist \circ llist\text{-}of\text{-}nellist)$
by (*simp add*: *map-fun-def nfusecat-def*)

lemma *not-null-lfusecat*:
$\neg lnull((lfusecat \circ (lmap\ llist\text{-}of\text{-}nellist \circ llist\text{-}of\text{-}nellist))\ nells)$
by (*simp add*: *llist-of-nellist.code*)

lemma *nfusecat-def3*:
$((llist\text{-}of\text{-}nellist \circ nfusecat)\ nells) =$
$((lfusecat \circ (lmap\ llist\text{-}of\text{-}nellist \circ llist\text{-}of\text{-}nellist))\ nells)$
proof −
let *?n2l* = $(lmap\ llist\text{-}of\text{-}nellist \circ llist\text{-}of\text{-}nellist)$
have *1*: $llist\text{-}of\text{-}nellist \circ nfusecat =$
$llist\text{-}of\text{-}nellist \circ nellist\text{-}of\text{-}llist \circ lfusecat \circ ?n2l$
by (*simp add*: *nfusecat-def2 rewriteL-comp-comp*)
have *2*: $\neg lnull((lfusecat \circ ?n2l)\ nells)$
using *not-null-lfusecat* **by** *auto*
have *3*: $(llist\text{-}of\text{-}nellist \circ nellist\text{-}of\text{-}llist \circ lfusecat \circ ?n2l)\ nells =$
$(lfusecat \circ ?n2l)\ nells$
using *2 nellist-of-llist-inverse* **by** *simp*
show *?thesis*
by (*metis 1 3*)
qed

lemma *nfusecat-NNil*[*simp*]:
$nfusecat\ (NNil\ nell) = nell$
apply *transfer*
by *simp*

lemma *nfusecat-NCons*[*simp*]:
nfusecat (*NCons nell nells*) = *nfuse nell* (*nfusecat nells*)
apply *transfer*
by (*simp add*: *lfuse-def llist.case-eq-if*)

lemma *nfusecat-nfirst*:
assumes (∃ *x1*. *nells* = *NNil x1*)
shows *nfusecat nells* = *nfirst nells*
proof −
obtain *nell* **where** *1*: *nells* = *NNil nell*
using *assms* **by** *auto*
have *2*: *nfusecat nells* = *nell*
by (*simp add*: *1*)
have *3*: *nfirst nells* = *nell*
by (*metis 1 ndropn-0 ndropn-nfirst nnth-NNil*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *nfusecat-NCons-nfirst*:
assumes (∀ *nell*. *nells* ≠ *NNil nell*)
shows *nfusecat nells* = *nfuse* (*nfirst nells*) (*nfusecat* (*ntl nells*))
by (*metis assms nellist-split-2-first nlast-NNil nfusecat-NCons not-le-imp-less ntaken-0*
ntaken-all ntaken-nlast zero-enat-def)

lemma *nfusecat-expand*:
nfusecat nells =
(*if is-NNil nells then nfirst nells else nfuse* (*nfirst nells*) (*nfusecat* (*ntl nells*)))
unfolding *is-NNil-def*
using *nfusecat-NCons-nfirst nfusecat-nfirst* **by** *simp blast*

lemma *nfusecat-expand-case*:
nfusecat nells = (*case nells of* (*NNil nell*) ⇒ *nell* |
(*NCons nell*′ *nells1*) ⇒ *nfuse nell*′ (*nfusecat nells1*))
by (*metis is-NNil-def nellist.case-eq-if nellist.collapse*(*2*) *nlast-NNil nfusecat-NCons nfusecat-NNil*)

lemma *nmap-nfusecat*:
nmap f (*nfusecat nells*) = (*nfusecat* (*nmap* (*nmap f*) *nells*))
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nmap f* (*nfusecat nells*) =
nmap f ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)
by (*simp add*: *nfusecat-def2*)
have *2*: *nmap f* ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*) =
nellist-of-llist (*lmap f* ((*lfusecat* ∘*?n2l*) *nells*))
using *nmap-nellist-of-llist not-null-lfusecat* **by** *auto*
have *3*: (*lmap f* ((*lfusecat* ∘ *?n2l*) *nells*)) =
lfusecat (*lmap* (*lmap f*) ((*lmap llist-of-nellist* ∘ *llist-of-nellist*) *nells*))

by (*simp add*: *lmap-lfusecat*)
have *4*: (*nfusecat* (*nmap* (*nmap f*) *nells*)) =
(*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*nmap* (*nmap f*) *nells*)
by (*simp add*: *nfusecat-def2*)
have *8*: (*lmap* (*lmap f*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
using *lmap-llist-of-nellist-nmap* **by** *blast*
show *?thesis*
using *1 2 3 4 8* **by** *fastforce*
qed

lemma *nfirst-nfuse-var*:
nfirst (*nfuse nellx nelly*) = *nfirst nellx*
by (*metis nfuse-def1 nlast-NNil ntaken-0 ntaken-nappend1 zero-enat-def zero-le*)

lemma *nfirst-nfusecat-nfirst*:
shows *nfirst*(*nfusecat nells*) = *nfirst*(*nfirst nells*)
proof (*cases nells*)
case (*NNil nell*)
then show *?thesis*
by (*simp add*: *nfusecat-expand*)
next
case (*NCons nell nells1*)
then show *?thesis*
proof −
have *1*: *nfusecat* (*NCons nell nells1*) = *nfuse nell* (*nfusecat nells1*)
by (*simp*)
have *2*: *nfirst* (*nfuse nell* (*nfusecat nells1*)) = *nfirst nell*
using *nfirst-nfuse-var* **by** *auto*
have *3*: *nfirst*(*nfirst* (*NCons nell nells1*)) = *nfirst nell*
by (*metis 1 nfirst-nfuse-var nfusecat-expand*)
show *?thesis*
using *1 2 3 NCons* **by** *fastforce*
qed
qed

lemma *nfirst-nfusecat*:
shows *nfirst*(*nfusecat* (*NCons nell nells*)) = *nfirst nell*
by (*simp add*: *nfirst-nfuse-var*)

lemma *nfusecat-nlength-eq-zero-conv*:
nlength (*nfusecat nells*) = *0* ⟷ (∀ *nell* ∈ *nset nells*. *nlength nell* = *0*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlength* (*nfusecat nells*) =
nlength ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)
by (*simp add*: *nfusecat-def2*)
have *2*: *nlength* ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*) =

epred (*llength* (*lfusecat* (*?n2l nells*)))
by *simp*
(*metis lbutlast-snoc llength-lbutlast nellist-of-llist-a-inverse nlength.rep-eq*)
have *3*: *nlength* (*nfusecat nells*) = *0* ⟷
epred (*llength* (*lfusecat* (*?n2l nells*))) = *0*
using *1 2* **by** *presburger*
have *4*: *llength* (*?n2l nells*) > *0*
by *simp*
have *5*: (*llength* (*lfusecat* (*?n2l nells*))) > *0*
by (*simp add*: *lfusecat-not-lnull-var*)
have *6*: *epred* (*llength* (*lfusecat* (*?n2l nells*))) = *0* ⟷
(*llength* (*lfusecat* (*lmap llist-of-nellist* (*llist-of-nellist nells*)))) = *1*
by (*metis 5 comp-apply epred-1 epred-inject not-iless0 zero-one-enat-neq*(*1*))
have *7*: (*llength* (*lfusecat* (*?n2l nells*))) ≤ *1* ⟷
(∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* ≤ *1*)
using *lfusecat-all-empty-or-LNil lset-lltl-llength-var* **by** *blast*
have *8*: (∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* > *0*)
by *simp*
have *9*: (*llength* (*lfusecat* (*?n2l nells*))) = *1* ⟷
(∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* = *1*)
by (*metis 5 7 8 ileI1 one-eSuc order-antisym-conv*)
have *10*: (∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* = *1*) ⟷
(∀ *nell* ∈ *nset nells*. *nlength nell* = *0*)
by *simp*
(*metis co.enat.exhaust-sel epred-1 llength-eq-0 llist-of-nellist-not-lnull nlength.rep-eq*
zero-neq-one)
show *?thesis* **using** *3 6 9 10* **by** *simp*
qed

lemma *is-NNil-nfusecat-a*:
assumes ∀ *i*. *i*≤*nlength nells* ⟶ *is-NNil* (*nnth nells i*)
shows *is-NNil*(*nfusecat nells*)
using *assms nfusecat-nlength-eq-zero-conv*[*of nells*]
by (*metis in-nset-conv-nnth is-NNil-def nle-le nlength-NNil ntaken-0 ntaken-all zero-enat-def*)

lemma *is-NNil-nfusecat-b*:
assumes *is-NNil*(*nfusecat nells*)
shows ∀ *i*. *i*≤*nlength nells* ⟶ *is-NNil* (*nnth nells i*)
using *assms nfusecat-nlength-eq-zero-conv*[*of nells*]
by (*metis in-nset-conv-nnth nellist.collapse*(*1*) *nellist.collapse*(*2*) *nlength-NCons nlength-NNil*
zero-ne-eSuc)

lemma *ntl-lfusecat* :
shows *ntl*(*nfusecat nells*) =
(*if is-NNil nells then ntl* (*nfirst nells*) *else*
(*if is-NNil* (*nfirst nells*) *then*
if is-NNil (*nfusecat* (*ntl nells*))
then ntl (*nfirst nells*)

```
        else ntl (nfusecat (ntl nells))
      else
        if is-NNil (nfusecat (ntl nells))
        then ntl (nfirst nells)
        else nappend (ntl (nfirst nells)) (ntl (nfusecat (ntl nells)))))

proof (cases nells)
case (NNil nell)
then show ?thesis by (metis nellist.disc(1) nfusecat-NNil nfusecat-expand)
next
case (NCons nell nells1)
then show ?thesis
  using nfusecat-NCons[of nell nells1] ntl-nfuse[of nell (nfusecat nells1)] by simp
    (metis ndropn-0 ndropn-nfirst nnth-0)
qed

lemma nfusecat-nappend:
assumes nfinite llx
shows nfusecat (nappend llx lly) = nfuse (nfusecat llx) (nfusecat lly)
proof −
 let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
 have 1: nfusecat (nappend llx lly) =
        ((nellist-of-llist ∘ lfusecat ∘ ?n2l) (nappend llx lly))
   by (simp add: nfusecat-def2)
 have 2: ((nellist-of-llist ∘ lfusecat ∘ ?n2l) (nappend llx lly)) =
        nellist-of-llist (lfusecat (?n2l (nappend llx lly)))
    by simp
 have 3: (llist-of-nellist (nappend llx lly)) =
        lappend (llist-of-nellist llx) (llist-of-nellist lly)
    by (simp add: llist-of-nellist-nappend)
 have 4: (lmap llist-of-nellist (lappend (llist-of-nellist llx) (llist-of-nellist lly))) =
         lappend (?n2l llx) (?n2l lly)
    using lmap-lappend-distrib by auto
 have 5: (lfusecat (lappend (?n2l llx) (?n2l lly))) =
        lfuse (lfusecat (?n2l llx)) (lfusecat (?n2l lly))
     using assms lfinite-lmap lfusecat-lappend nfinite-def by (metis comp-def)
 have 6: nellist-of-llist (lfuse (lfusecat (?n2l llx)) (lfusecat (?n2l lly))) =
        nfuse (nfusecat llx) (nfusecat lly)
    by (simp add: llist-of-nellist.code nfuse.abs-eq nfusecat-def2)
 show ?thesis
 using 1 2 3 4 5 6 by simp
qed

lemma nfusecat-split:
 assumes (Suc n) ≤ nlength nells
shows nfusecat nells = nfuse (nfusecat (ntaken n nells)) (nfusecat (ndropn (Suc n) nells))
using assms
by (metis nappend-ntaken-ndropn nfinite-ntaken nfusecat-nappend)

lemma nfusecat-split-1:
```

assumes $(Suc\ n) \leq nlength\ nells$
shows $nfusecat\ nells = nfuse\ (nfusecat\ (ntake\ n\ nells))\ (nfusecat\ (ndropn\ (Suc\ n)\ nells))$
using *assms*
by (*simp add*: *nfusecat-split ntake-eq-ntaken*)

lemma *nfuse-nfinite*:
shows $nfinite\ (nfuse\ nellx\ nelly) \longleftrightarrow nfinite\ nellx \wedge nfinite\ nelly$
apply *transfer*
using *lfuse-lfinite* **by** *blast*

lemma *nfusecat-nfinite*:
assumes $\forall\, nell \in nset\ nells.\ nlength\ nell > 0$
$\forall\, nell \in nset\ nells.\ nfinite\ nell$
$nlastnfirst\ nells$
shows $nfinite\ (nfusecat\ nells) \longleftrightarrow nfinite\ nells$
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: $nfinite\ (nfusecat\ nells) \longleftrightarrow$
$nfinite\ (((nellist\text{-}of\text{-}llist \circ lfusecat \circ ?n2l)\ nells))$
by (*simp add*: *nfusecat-def2*)
have *2*: $nfinite\ (((nellist\text{-}of\text{-}llist \circ lfusecat \circ ?n2l)\ nells)) \longleftrightarrow$
$lfinite\ (llist\text{-}of\text{-}nellist\ (((nellist\text{-}of\text{-}llist \circ lfusecat \circ ?n2l)\ nells)))$
using *nfinite-def* **by** *blast*
have *3*: $lfinite\ (llist\text{-}of\text{-}nellist\ (((nellist\text{-}of\text{-}llist \circ lfusecat \circ ?n2l)\ nells))) \longleftrightarrow$
$lfinite\ (lfusecat\ (?n2l\ nells))$
by (*simp add*: *lbutlast-lfinite*)
have *4*: $\forall\, nell \in lset\ (llist\text{-}of\text{-}nellist\ nells).\ nlength\ nell > 0$
using *assms*(*1*) **by** *force*
have *5*: $\forall\, nell \in lset\ (llist\text{-}of\text{-}nellist\ nells).\ epred(llength\ (llist\text{-}of\text{-}nellist\ nell)) > 0$
using *4* **by** *fastforce*
have *6*: $\forall\, nell \in lset\ (llist\text{-}of\text{-}nellist\ nells).\ \neg\ lnull\ (ltl\ (llist\text{-}of\text{-}nellist\ nell))$
by (*metis 5 epred-llength less-numeral-extra*(*3*) *llength-LNil llist.collapse*(*1*))
have *7*: $\forall\, nell \in lset\ (?n2l\ nells).\ \neg\ lnull\ (ltl\ nell)$
by (*simp add*: *6*)
have *8*: $\forall\, nell \in lset\ (?n2l\ nells).\ lfinite\ nell$
using *assms*(*2*) *nfinite-def* **by** *fastforce*
have *9*: $llastlfirst\ (?n2l\ nells)$
using *assms*(*3*) *nlastnfirst.rep-eq* **by** *auto*
have *10*: $lfinite\ (lfusecat\ (?n2l\ nells)) \longleftrightarrow lfinite\ (?n2l\ nells)$
using *7 8 9 lfusecat-lfinite* **by** *blast*
have *11*: $lfinite\ (?n2l\ nells) \longleftrightarrow nfinite\ nells$
by (*simp add*: *nfinite-def*)
show *?thesis* **using** *1 2 3 10 11* **by** *presburger*
qed

lemma *nlastfirst-nfusecat-nlast*:
assumes *nfinite nells*
$nlastnfirst\ nells$
$\forall\, nell \in nset\ nells.\ nfinite\ nell$

shows *nlast*(*nfusecat nells*) = *nlast*(*nlast nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlast*(*nfusecat nells*) =
nlast (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))
by (*simp add*: *nfusecat-def2*)
have *2*: *nlast* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)) =
llast (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)))
by (*metis* (*no-types*, *lifting*) *llast-linfinite nfinite-def nlast-llast nlast-not-nfinite*)
have *3*: *llast* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))) =
llast (*lfusecat* (*?n2l nells*))
by (*metis* (*no-types*, *lifting*) *2 comp-def lbutlast.disc-iff*(*1*) *nellist-of-llist-a-inverse*
nlast-nellist-of-llist-a-lnull not-lnull-eq-lappend-lbutlast-llast)
have *4*: *lfinite* (*?n2l nells*)
using *assms*(*1*) *lfinite-lmap nfinite-def* **by** *auto*
have *5*: ¬*lnull* (*?n2l nells*)
by *simp*
have *6*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬*lnull nell*
by *simp*
have *7*: *llastlfirst* (*?n2l nells*)
using *assms*(*2*) *nlastnfirst.rep-eq* **by** *auto*
have *8*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*3*) *nfinite-def* **by** *auto*
have *9*: *llast* (*lfusecat* (*?n2l nells*)) = *llast*(*llast* (*?n2l nells*))
using *4 5 6 7 8 llastfirst-lfusecat-llast* **by** *blast*
have *10*: *llast*(*llast* (*?n2l nells*)) = *nlast* (*nlast nells*)
by (*metis assms*(*1*) *assms*(*3*) *comp-apply llast-lmap llist-of-nellist-not-lnull nfinite-def*
nlast-llast nset-nlast)
show *?thesis* **using** *1 2 3 9 10* **by** *presburger*
qed

lemma *nfusecat-nlength-nfinite*:
assumes *nfinite nells*
∀ *nell* ∈ *nset nells*. *nfinite nell*
nlastnfirst nells
shows *nlength*(*nfusecat nells*) =
(∑ *i* = *0* .. (*the-enat*((*nlength nells*))) . (*nlength* (*nnth nells i*)))
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlength*(*nfusecat nells*) = *nlength* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))
by (*simp add*: *nfusecat-def2*)
have *2*: *nlength* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)) =
epred (*llength* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))))
using *nlength.rep-eq* **by** *blast*
have *3*: *epred* (*llength* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)))) =
epred (*llength* (*lfusecat* (*?n2l nells*)))
using *one-eSuc plus-1-eSuc*(*2*) **by** *simp*
have *4*: *lfinite* (*?n2l nells*)
using *assms*(*1*) *lfinite-lmap nfinite-def* **by** *auto*

have *5*: ¬*lnull* (*?n2l nells*)
by *simp*
have *6*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬ *lnull nell*
by *simp*
have *7*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*2*) *nfinite-def* **by** *auto*
have *8*: *llastlfirst* (*?n2l nells*)
using *assms*(*3*) *nlastnfirst.rep-eq* **by** *auto*
have *9*: *epred* (*llength* (*lfusecat* (*?n2l nells*))) =
epred (*eSuc*(∑ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .
epred(*llength* (*lnth* (*?n2l nells*) *i*))))
by (*metis 4 5 6 7 8 lfusecat-llength-lfinite*)
have *10*: *epred* (*eSuc*(∑ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .
epred(*llength* (*lnth* (*?n2l nells*) *i*)))) =
((∑ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .
epred(*llength* (*lnth* (*?n2l nells*) *i*))))
using *epred-eSuc* **by** *blast*
have *11*: ((*epred*(*llength* (*?n2l nells*)))) = ((*nlength nells*))
by *simp*
have *12*: ⋀*j*. *j* ≤ ((*epred*(*llength* (*?n2l nells*))))⟶
epred(*llength* (*lnth* (*?n2l nells*) *j*)) = *nlength* (*nnth nells j*)
by (*metis 5 co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llength-llist-of-nellist*
llength-lmap lnth-llist-of-nellist lnth-lmap min-def the-enat.simps)
have *13*: ((∑ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .
epred(*llength* (*lnth* (*?n2l nells*) *i*)))) =
((∑ *i* = *0* .. (*the-enat*(*nlength nells*)). *nlength* (*nnth nells i*)))
using *12 assms*(*1*) *nfinite-nlength-enat* **by** *force*
show *?thesis* **using** *1 2 3 9 10 13* **by** *metis*
qed

lemma *nlastnfirst-ntaken*:
assumes *n*≤*nlength nells*
nlastnfirst nells
shows *nlastnfirst* (*ntaken n nells*)
using *assms*
by (*metis Suc-ile-eq antisym-conv2 nappend-ntaken-ndropn nfinite-ntaken nlastnfirst-nappend-nfinite*
ntaken-all)

lemma *nlastnfirst-ntake*:
assumes *n*≤*nlength nells*
nlastnfirst nells
shows *nlastnfirst* (*ntake n nells*)
proof (*cases n*)
case (*enat nat*)
then show *?thesis* **by** (*metis assms nlastnfirst-ntaken ntake-eq-ntaken*)
next
case *infinity*

then show *?thesis*
by (*simp add*: *assms ntake-all*)
qed

lemma *nfusecat-ntake*:
assumes (*enat n*) ≤ *nlength nells*
∀ *nell* ∈ *nset nells*. *nfinite nell*
nlastnfirst nells
shows *nfusecat* (*ntake n nells*) =
ntake (((∑ *i* = *0* .. *n* . (*nlength* (*nnth nells i*)))))
(*nfusecat nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nfusecat* (*ntake n nells*) =
(((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*ntake n nells*)))
by (*simp add*: *nfusecat-def2*)
have *2*: (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*ntake n nells*))) =
nellist-of-llist (*lfusecat* (*?n2l* (*ntake n nells*)))
by *simp*
have *3*: *llist-of-nellist*(*ntake n nells*) = *ltake* (*eSuc n*) (*llist-of-nellist nells*)
by (*metis co.enat.distinct*(*1*) *llist-of-nellist-inverse-b llist-of-nellist-not-lnull*
ltake.disc(*2*) *nellist-of-llist-inverse ntake.abs-eq*)
have *4*: (*?n2l* (*ntake n nells*)) = *ltake* (*eSuc n*) (*?n2l nells*)
using *3* **by** *auto*
have *5*: ¬ *lnull* (*?n2l nells*)
by *simp*
have *6*: (*Suc n*) ≤ *llength* (*?n2l nells*)
by (*metis 5 Suc-ile-eq assms*(*1*) *co.enat.collapse comp-apply iless-Suc-eq llength-eq-0 llength-lmap*
nlength.rep-eq)
have *7*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬ *lnull nell*
by *simp*
have *8*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*2*) *nfinite-def* **by** *fastforce*
have *9*: *llastlfirst* (*?n2l nells*)
using *assms*(*3*) *nlastnfirst.rep-eq* **by** *auto*
have *10*: (*lfusecat* (*ltake* (*eSuc n*) (*?n2l nells*))) =
ltake (*if* (*Suc n*) =*0 then 0 else eSuc*(∑ *i* = *0* .. *n* .
epred(*llength* (*lnth* (*?n2l nells*) *i*))))
(*lfusecat* (*?n2l nells*))
using *lfusecat-ltake*[*of* (*?n2l nells*) *Suc n*]
5 6 7 8 9 diff-Suc-1 eSuc-enat **by** *presburger*
have *11*: (*if* (*Suc n*) =*0 then 0 else eSuc*(∑ *i* = *0* .. *n* .
epred(*llength* (*lnth* (*?n2l nells*) *i*)))) =
(*eSuc*(∑ *i* = *0* .. *n* . *epred*(*llength* (*lnth* (*?n2l nells*) *i*))))
using *nat.simps*(*3*) **by** *presburger*
have *111*: ⋀*j*. *j* ≤*n* ⟶ (*min* (*enat j*) (*epred* (*llength* (*llist-of-nellist nells*)))) = *j*
by (*metis assms*(*1*) *enat-ord-simps*(*1*) *llength-llist-of-nellist min.bounded-iff min-def*)
have *12*: ⋀ *j*. *j* ≤*n* ⟶
epred(*llength* (*lnth* (*?n2l nells*) *j*)) = *nlength* (*nnth nells j*)

using *5 111 assms lnth-llist-of-nellist*[*of nells*]
by (*metis* (*full-types*) *co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lmap lnth-lmap min.order-iff nlength.rep-eq o-apply the-enat.simps*)
have *13*: (*lfusecat* (*?n2l nells*)) = *llist-of-nellist* (*nfusecat nells*)
by (*simp add*: *lfusecat-not-lnull-var nfusecat-def2*)
have *14*: *ltake* (*if* (*Suc n*) =*0 then 0 else eSuc*(∑ *i* = *0* .. *n* .
epred(*llength* (*lnth* (*?n2l nells*) *i*))))
(*lfusecat* (*?n2l nells*)) =
ltake (*eSuc*(∑ *i* = *0* .. *n* . *nlength* (*nnth nells i*))) (*llist-of-nellist* (*nfusecat nells*))
using *12 13* **by** *auto*
have *15*: *nellist-of-llist* (*ltake* (*eSuc*(∑ *i* = *0* .. *n* .
nlength (*nnth nells i*))) (*llist-of-nellist* (*nfusecat nells*))) =
ntake ((∑ *i* = *0* .. *n* . *nlength* (*nnth nells i*))) (*nfusecat nells*)
by (*metis llist-of-nellist-inverse-b llist-of-nellist-not-lnull ntake.abs-eq*)
show *?thesis*
by (*metis 10 14 15 2 4 nfusecat-def2*)
qed

lemma *nfusecat-ndropn*:
assumes *n* < *nlength nells*
∀ *nell* ∈ *nset nells*. *nfinite nell*
nlastnfirst nells
shows *nfusecat* (*ndropn n nells*) =
ndropn (*the-enat*(*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *nlength*(*nnth nells i*))))
(*nfusecat nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nfusecat* (*ndropn n nells*) =
(((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*ndropn n nells*)))
by (*simp add*: *nfusecat-def2*)
have *2*: (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*ndropn n nells*))) =
nellist-of-llist (*lfusecat* (*lmap llist-of-nellist* (*ldropn n* (*llist-of-nellist nells*))))
using *assms* **by** *simp*
have *4*: *nellist-of-llist* (*lfusecat* (*lmap llist-of-nellist* (*ldropn n* (*llist-of-nellist nells*)))) =
nellist-of-llist (*lfusecat* (*lmap llist-of-nellist* (*ldrop n* (*llist-of-nellist nells*))))
by (*simp add*: *ldrop-enat*)
have *5*: (*lmap llist-of-nellist* (*ldrop n* (*llist-of-nellist nells*))) = *ldrop n* (*?n2l nells*)
by (*simp*)
have *6*: ¬ *lnull* (*?n2l nells*)
by *simp*
have *7*: *n* < *llength* (*?n2l nells*)
by (*metis 5 assms*(*1*) *ldrop-enat llist.map-disc-iff llist-of-nellist-ndropn llist-of-nellist-not-lnull lnull-ldrop min-def nlength.rep-eq not-le-imp-less order-less-imp-le the-enat.simps*)
have *8*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬ *lnull nell*
by *simp*
have *9*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*2*) *nfinite-def* **by** *fastforce*
have *10*: *llastlfirst* (*?n2l nells*)
using *assms*(*3*) *nlastnfirst.rep-eq* **by** *auto*
have *11*: *lfusecat* (*ldrop* (*enat n*) (*?n2l nells*)) =

ldrop (*if n = 0 then 0 else* ($\sum$ *i = 0 .. (n−1)* .
epred(*llength* (*lnth* (*?n2l nells*) *i*))))
(*lfusecat* (*?n2l nells*))
using *10 6 7 8 9 lfusecat-ldrop* **by** *blast*
have *111*: $\bigwedge j$. *0< j* $\wedge$ *j <n−1* $\longrightarrow$
(*min* (*enat j*) (*epred* (*llength* (*llist-of-nellist nells*)))) = *j*
by (*metis 7 comp-apply epred-enat epred-min less-imp-of-nat-less llength-lmap min.absorb3*
min-less-iff-conj of-nat-eq-enat)
have *112*: *n−1 < llength* (*?n2l nells*)
by (*metis 7 One-nat-def Suc-ile-eq Suc-pred epred-0 epred-enat not-gr-zero order-less-imp-le zero-enat-def*)

have *12*: $\bigwedge j$. *0< n* $\wedge$ *j < n−1* $\longrightarrow$
epred(*llength* (*lnth* (*?n2l nells*) *j*)) = *nlength*(*nnth nells j*)
using *6 7 111 lnth-llist-of-nellist*[*of nells*]
by *simp*
(*metis canonically-ordered-monoid-add-class.lessE diff-le-self dual-order.strict-trans1*
enat-less-enat-plusI enat-min-eq-0-iff lnth-lmap nlength.rep-eq not-gr-zero the-enat.simps
zero-enat-def)
have *13*: (*lfusecat* (*?n2l nells*)) = *llist-of-nellist* (*nfusecat nells*)
by (*simp add*: *lfusecat-not-lnull-var nfusecat-def2*)
have *131*: $\bigwedge j$. *j< n* $\Longrightarrow$ (*min* (*enat j*) (*epred* (*llength* (*llist-of-nellist nells*)))) =*j*
by (*metis* (*full-types*) *assms*(*1*) *min.assoc min.orderE min-enat-simps*(*1*) *nlength.rep-eq*
order-less-imp-le)
have *132*: (*if n = 0 then 0 else* $\sum$ *i = 0..n − 1. epred* (*llength* (*lnth* (*?n2l nells*) *i*))) =
(*if n = 0 then 0 else* ($\sum$ *i = 0 .. (n−1)* . *nlength*(*nnth nells i*)))
using *sum.cong*[*of* {*0..n−1*} {*0..n−1*} λ*i. epred* (*llength* (*lnth* (*?n2l nells*) *i*))
λ*i. nlength*(*nnth nells i*)] *assms 7 131 12 lnth-llist-of-nellist*[*of nells*]
by (*metis 112 atLeastAtMost-iff comp-apply diff-less less-numeral-extra*(*1*) *llength-lmap*
lnth-lmap nlength.rep-eq not-gr-zero order-neq-le-trans the-enat.simps)
have *14*: *ldrop* (*if n = 0 then 0 else* ($\sum$ *i = 0 .. (n−1)* . *epred*(*llength* (*lnth* (*?n2l nells*) *i*))))
(*lfusecat* (*?n2l nells*)) =
ldrop (*if n = 0 then 0 else* ($\sum$ *i = 0 .. (n−1)* . *nlength*(*nnth nells i*))) (*llist-of-nellist* (*nfusecat*
nells))
using *13 132* **by** *presburger*
have *15*: *0<n* $\Longrightarrow$ ($\bigwedge$ *j*. *j* $\leq$*n−1* $\longrightarrow$ *epred*(*llength* (*lnth* (*?n2l nells*) *j*)) < ∞)
using *7 9*
by (*metis* (*no-types*, *opaque-lifting*) *112 enat-ord-simps*(*1*) *enat-ord-simps*(*4*) *epred-llength*
in-lset-conv-lnth lfinite-ltl llength-eq-infty-conv-lfinite order-le-less-trans)
have *16*: *0< n* $\Longrightarrow$ (($\sum$ *i = 0 .. (n−1)* . *epred*(*llength* (*lnth* (*?n2l nells*) *i*)))) < ∞
using *15*
proof (*induct n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases n=0*)
case *True*
then show *?thesis* **using** *Suc* **by** *simp*
next

```
      case False
      then show ?thesis using Suc
      by simp
        (metis One-nat-def Suc-pred' dual-order.refl plus-enat-simps(1) sum.atLeast0-atMost-Suc)
      qed
    qed
  have 17: ∃ m. (enat m) = (if n = 0 then 0 else (∑ i = 0 .. (n−1) .
          epred(llength (lnth (?n2l nells) i))))
    by (metis 16 less-infinityE not-gr-zero zero-enat-def)
  obtain m where 18: (enat m) = (if n = 0 then 0 else
                    (∑ i = 0 .. (n−1) . epred(llength (lnth (?n2l nells) i))))
    using 17 by blast
  have 19: ldrop (if n = 0 then 0 else
                  (∑ i = 0 .. (n−1) . epred(llength (lnth (?n2l nells) i))))
              (lfusecat (?n2l nells)) =
            ldropn m (lfusecat (?n2l nells))
     using 18 ldrop-enat[of m (lfusecat (?n2l nells))]
       by presburger
  have 20:  enat m ≤ epred (llength (lfusecat (?n2l nells)))
      by (metis (no-types, lifting) 11 19 6 7 8 co.enat.exhaust-sel iless-Suc-eq in-lset-ldropD
        lfusecat-not-lnull-var linorder-not-le llength-eq-0 lnull-ldrop lnull-ldropn)
  have 21: nellist-of-llist (ldropn m (lfusecat (?n2l nells))) = ndropn m (nfusecat nells)
      using 20
      by (metis 13 llist-of-nellist-inverse-b llist-of-nellist-not-lnull min-def ndropn.abs-eq
        the-enat.simps)
  have 22: m = (the-enat(if n = 0 then 0 else (∑ i = 0 .. (n−1) . nlength(nnth nells i))))
     by (metis (mono-tags, lifting) 132 18 the-enat.simps)
  show ?thesis using 1 2 4 5 11 19 21 22 by (metis)
qed

lemma nridx-nfusecat-ntake:
 assumes nridx R (nfusecat nells)
         (enat n) ≤ nlength nells
         nlastnfirst nells
         ∀ nell ∈ nset nells. nfinite nell
 shows    nridx R (nfusecat (ntake n nells))
using assms by (metis nfusecat-ntake nle-le nridx-ntake-a ntake-all)

lemma nridx-nfusecat-ndrop:
 assumes nridx R (nfusecat nells)
         (enat n) < nlength nells
         nlastnfirst nells
         ∀ nell ∈ nset nells. nfinite nell
 shows    nridx R (nfusecat (ndropn n nells))
proof −
let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
 have 1: nridx R (nfusecat (ndropn n nells)) ⟷
         nridx R ((nellist-of-llist ∘ lfusecat ∘ ?n2l) (ndropn n nells))
   by (simp add: nfusecat-def2)
 have 2:  nridx R ((nellist-of-llist ∘ lfusecat ∘ ?n2l) (ndropn n nells)) ⟷
```

ridx R (llist-of-nellist ((nellist-of-llist ∘ lfusecat ∘ ?n2l) (ndropn n nells)))
using *nridx.rep-eq* **by** *blast*
have *3*: *ridx R (llist-of-nellist ((nellist-of-llist ∘ lfusecat ∘ ?n2l) (ndropn n nells))) ⟷*
ridx R (((lfusecat ∘ ?n2l) (ndropn n nells)))
using *not-null-lfusecat*
by (*metis* (*no-types*, *lifting*) *comp-apply nridx.abs-eq nridx.rep-eq*)
have *4*: *llastlfirst (?n2l nells)*
using *assms nlastnfirst.rep-eq* **by** *auto*
have *5*: *∀ nell ∈ lset (?n2l nells). lfinite nell*
using *assms nfinite-def* **by** *fastforce*
have *6*: *(((lfusecat ∘ ?n2l) (ndropn n nells))) =*
lfusecat (lmap llist-of-nellist (ldropn n (llist-of-nellist nells)))
by (*simp add*: *assms*(*2*))
have *7*: (*lmap llist-of-nellist (ldropn n (llist-of-nellist nells))) =*
ldropn n (lmap llist-of-nellist (llist-of-nellist nells))
by *auto*
have *8*: *ridx R (lfusecat (?n2l nells))*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *comp-apply nfusecat-def2 nridx.abs-eq ridx-expand-1*)
have *9*: *enat n < llength (?n2l nells)*
by (*metis assms*(*2*) *co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llength-lmap*
llist-of-nellist-not-lnull nlength.rep-eq order-less-imp-le)
have *10*: *ridx R (lfusecat (ldropn n (?n2l nells)))*
using *ridx-lfusecat-ldrop*[*of R (?n2l nells) n*]
by (*metis* (*mono-tags*, *lifting*) *4 5 8 9 comp-apply in-lset-conv-lnth ldrop-enat llength-lmap*
llist-of-nellist-not-lnull lnth-lmap)
show *?thesis*
using *1 10 2 3 6 7* **by** (*metis comp-apply*)
qed

lemma *nridx-nfusecat*:
assumes *nlastnfirst nells*
∀ nell ∈ nset nells. 0 < nlength nell
∀ nell ∈ nset nells. nfinite nell
shows *nridx R (nfusecat nells) ⟷ (∀ i. i ≤ nlength nells ⟶ nridx R (nnth nells i))*
proof −
let *?n2l* = (*lmap llist-of-nellist ∘ llist-of-nellist*)
have *1*: *nridx R (nfusecat nells) ⟷*
nridx R ((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells)
by (*simp add*: *nfusecat-def*)
have *2*: *nridx R ((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells) ⟷*
ridx R (llist-of-nellist ((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells))
using *nridx.rep-eq* **by** *blast*
have *3*: *ridx R (llist-of-nellist ((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells)) ⟷*
ridx R (((lfusecat ∘ ?n2l) nells))
using *not-null-lfusecat nridx.abs-eq* **by** *fastforce*
have *4*: *llastlfirst (?n2l nells)*
using *assms*(*1*) *nlastnfirst.rep-eq* **by** *auto*
have *5*: *∀ nell ∈ lset (?n2l nells). 1 < llength nell*
by (*metis assms*(*2*) *epred-1 comp-apply imageE less-numeral-extra*(*3*) *llist.set-map*

llist-of-nellist-not-lnull lset-llist-of-nellist-a nlength.rep-eq not-lnull-llength order-neq-le-trans)
have *6*: $\forall$ *nell* $\in$ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*3*) *nfinite-def* **by** *fastforce*
have *7*: *ridx R* (((*lfusecat* $\circ$ *?n2l*) *nells*)) $\longleftrightarrow$
($\forall$ *i*. *i* < *llength* (*?n2l nells*) $\longrightarrow$ *ridx R* (*lnth* (*?n2l nells*) *i*))
using *4 5 6 ridx-lfusecat* **by** *fastforce*
have *8*: *epred*(*llength* (*?n2l nells*)) = *nlength nells*
by *simp*
have *9*: $\bigwedge$*j*. *j* < *llength* (*?n2l nells*) $\longrightarrow$
(*lnth* (*?n2l nells*) *j*) = *llist-of-nellist*(*nnth nells j*)
by *simp*
(*metis eSuc-epred eSuc-ile-mono gr-implies-not-zero ileI1 lnth-llist-of-nellist min-def the-enat.simps*)
have *10*: ($\forall$ *i*. *i* < *llength* (*?n2l nells*) $\longrightarrow$ *ridx R* (*lnth* (*?n2l nells*) *i*)) $\longleftrightarrow$
($\forall$ *i*. *i* $\leq$ *nlength* (*nells*) $\longrightarrow$ *nridx R* (*nnth nells i*))
by (*metis* (*mono-tags*, *lifting*) *8 9 co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llist.map-disc-iff llist-of-nellist-not-lnull nridx.rep-eq*)
show *?thesis* **using** *1 2 3 10 7* **by** *blast*
qed

lemma *nfusecat-split-nsubn*:
assumes *nlastnfirst nells*
$\forall$ *nell* $\in$ *nset nells*. 0 < *nlength nell*
$\forall$ *nell* $\in$ *nset nells*. *nfinite nell*
shows ($\forall$ *i*. *i*$\leq$ *nlength nells* $\longrightarrow$ *nridx* (λ *a b*. *f* (*nsubn* σ *a b*)) (*nnth nells i*)) $\longleftrightarrow$
nridx (λ *a b*. *f* (*nsubn* σ *a b*)) (*nfusecat nells*)
using *assms nridx-nfusecat* **by** *blast*

lemma *nidx-nfusecat*:
assumes *nlastnfirst nells*
$\forall$ *nell* $\in$ *nset nells*. 0 < *nlength nell*
$\forall$ *nell* $\in$ *nset nells*. *nfinite nell*
shows *nidx* (*nfusecat nells*) $\longleftrightarrow$ ($\forall$ *i*. *i* $\leq$ *nlength nells* $\longrightarrow$ *nidx* (*nnth nells i*))
using *assms nridx-nfusecat nridx-nidx* **by** *blast*

lemma *nnth-sum-expand*:
assumes *nlastnfirst nells*
$\forall$ *nell* $\in$ *nset nells*. 0 < *nlength nell*
$\forall$ *nell* $\in$ *nset nells*. *nfinite nell*
(*enat i*) $\leq$ *nlength nells*
nfinite nells
shows ($\sum$ *i* = 0 .. (*the-enat*((*nlength nells*))) . (*nlength* (*nnth nells i*))) =
(*nlength* (*nnth nells i*)) +
($\sum$ *j* $\in$ {*k*. *k*$\neq$*i* $\wedge$ *k*$\leq$ (*the-enat*((*nlength nells*)))} . (*nlength* (*nnth nells j*)))
proof −
have *1*: {*k*. *k*$\leq$ (*the-enat*((*nlength nells*)))} = *insert i* {*k*. *k*$\neq$*i* $\wedge$ *k*$\leq$ (*the-enat*((*nlength nells*)))}
using *assms* **by** *auto*
(*metis enat-ord-simps*(*1*) *nfinite-nlength-enat the-enat.simps*)
have *2*: {*0*.. (*the-enat*((*nlength nells*)))} = {*k*. *k*$\leq$ (*the-enat*((*nlength nells*)))}

```
   by auto
 have 3: (∑ i = 0 .. (the-enat((nlength nells))) . (nlength (nnth nells i))) =
        (∑ i ∈{k. k≤ (the-enat((nlength nells)))} . (nlength (nnth nells i)))
      using 2 by presburger
 have 4: (∑ i ∈{k. k≤ (the-enat((nlength nells)))} .(nlength (nnth nells i))) =
        (∑ j ∈(insert i {k. k≠i ∧ k≤ (the-enat((nlength nells)))}) . (nlength (nnth nells j)))
     using 1 by presburger
 have 5: (∑ j ∈(insert i {k. k≠i ∧ k≤ (the-enat((nlength nells)))}) . (nlength (nnth nells j))) =
        (nlength (nnth nells i)) +
       (∑ j ∈ {k. k≠i ∧ k≤ (the-enat((nlength nells)))} . (nlength (nnth nells j)))
    by auto
 show ?thesis
 using 3 4 5 by presburger
qed

lemma nfusecat-nlength-a:
 assumes nlastnfirst nells
        ∀ nell ∈ nset nells. 0 < nlength nell
        ∀ nell ∈ nset nells. nfinite nell
         i ≤ nlength nells
        (enat j) ≤ nlength (nnth nells i)
 shows (enat j) ≤ nlength (nfusecat nells)
proof (cases nfinite nells)
case True
then show ?thesis
  proof −
   have 2: nlength (nfusecat nells) =
          (∑ i = 0 .. (the-enat((nlength nells))) . (nlength (nnth nells i)))
    using True assms nfusecat-nlength-nfinite by fastforce
   have 3: nlength (nnth nells i) ≤
          (∑ i = 0 .. (the-enat((nlength nells))) . (nlength (nnth nells i)))
     using nnth-sum-expand[of nells i] assms
     by (metis (no-types, lifting) True le-iff-add)
   show ?thesis using assms 2 3 by force
 qed
next
case False
then show ?thesis
  proof −
   have 5: ¬ nfinite (nfusecat nells)
     using False assms nfusecat-nfinite by blast
   show ?thesis
   using 5
   by (simp add: nfinite-conv-nlength-enat)
  qed
qed
```

1.3.10 nkfilter

lemma *nkfilter-def1*:

assumes ($\exists$ $x \in$ *nset nell*. *P x*)
shows *nkfilter P n nell* = *nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell* (*niterates Suc n*)))
using *assms*
apply *transfer*
by (*simp add*: *kfilter-def*)
(*metis imageE llist.set-map lmap-fst-lzip*)

lemma *nmap-fst-nzip-a*:
assumes *nlength xs* = *nlength ys*
shows *nmap fst* (*nzip xs ys*) = *xs*
using *assms*
apply *transfer*
by *simp*
(*meson co.enat.expand llength-eq-0 lmap-fst-lzip-a*)

lemma *nmap-snd-nzip*:
assumes *nlength xs* = *nlength ys*
shows *nmap snd* (*nzip xs ys*) = *ys*
using *assms*
apply *transfer*
by *simp*
(*meson co.enat.expand llength-eq-0 lmap-snd-lzip*)

lemma *nmap-fst-nzip*:
(*nmap fst* (*nzip nell* (*niterates Suc n*))) = *nell*
apply *transfer*
by (*simp add*: *lmap-fst-lzip*)

lemma *nfilter-nkfilter-1*:
assumes ($\exists$ $x \in$ *nset nell*. *P x*)
shows (*nmap fst* (*nfilter* (*P*∘*fst*) (*nzip nell* (*niterates Suc n*)))) =
(*nfilter P nell*)
using *assms*
by (*metis nfilter-nmap nmap-fst-nzip*)

lemma *nkfilter-NNil* [*simp*]:
shows *P b* $\Longrightarrow$ *nkfilter P n* (*NNil b*) = (*NNil n*)
by *transfer auto*

lemma *nkfilter-NCons* [*simp*]:
assumes ($\exists$ $x \in$ *nset nell*. *P x*)
shows *nkfilter P n* (*NCons x nell*) =
(*if P x then NCons n* (*nkfilter P* (*Suc n*) *nell*) *else nkfilter P* (*Suc n*) *nell*)
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv*)

lemma *nkfilter-NCons-a* [*simp*]:
assumes ¬($\exists$ $x \in$ *nset nell*. *P x*)

$P\ x$
shows *nkfilter P n (NCons x nell) = (NNil n)*
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv*)

lemma *nkfilter-nlength*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows *nlength(nkfilter P n nell) = nlength(nfilter P nell)*
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv kfilter-llength*)

lemma *nkfilter-upperbound*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ i \leq n + nlength\ nell$
using *assms*
by *transfer*
(*auto*,
simp add: *kfilter-lnull-conv*,
metis co.enat.exhaust-sel iadd-Suc-right iless-Suc-eq kfilter-upperbound llength-eq-0)

lemma *nkfilter-lowerbound*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $n \leq nnth\ (nkfilter\ P\ n\ nell)\ i$
using *assms*
by *transfer*
(*auto*,
metis co.enat.collapse iless-Suc-eq kfilter-lnth-n-zero-a llength-eq-0)

lemma *nkfilter-mono*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$(Suc\ i) \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ i < nnth\ (nkfilter\ P\ n\ nell)\ (Suc\ i)$
using *assms*
by *transfer*
(*auto*,
metis diff-Suc-1 eSuc-epred epred-enat epred-le-epredI lidx-kfilter-gr iless-Suc-eq leD lessI
llength-eq-0 min.cobounded1 min-def the-enat.simps)

lemma *nkfilter-nfilter*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $(nnth\ nell\ ((nnth\ (nkfilter\ P\ n\ nell)\ i) - n)) = (nnth\ (nfilter\ P\ nell)\ i)$
using *assms*
apply *transfer*
proof (*auto simp add*: *kfilter-lnull-conv split*:*if-split-asm*)
fix *nella* :: $'a\ llist$ **and** *Pa* :: $'a \Rightarrow bool$ **and** *ia* :: *nat* **and** *na* :: *nat* **and** *x* :: $'a$

assume *a1*: $x \in lset\ nella$
assume *a2*: $Pa\ x$
assume *a3*: $enat\ ia \leq epred\ (llength\ (kfilter\ Pa\ na\ nella))$
assume *a4*: $\neg\ lnull\ nella$
have *f5*: $\forall\ e\ ea.\ \neg\ (e::enat) \leq ea \vee min\ e\ ea = e$
by (*simp add*: *min-def-raw*)
have *f6*: $\forall\ n\ p\ na\ as.\ \neg\ enat\ n < llength\ (kfilter\ p\ na\ (as::'a\ llist)) \vee$
$enat\ (lnth\ (kfilter\ p\ na\ as)\ n) < enat\ na + llength\ as$
by (*meson kfilter-upperbound*)
have *f7*: $min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))) = enat\ ia$
using *f5 a3* **by** *blast*
then have *f8*: $(enat\ ia < llength\ (kfilter\ Pa\ 0\ nella)) =$
$(enat\ (the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))) <$
$llength\ (kfilter\ Pa\ na\ nella))$
by (*simp add*: *kfilter-llength*)
have *f9*: $\forall\ n\ p\ na\ as.\ \neg\ enat\ n < llength\ (kfilter\ p\ na\ (as::'a\ llist)) \vee$
$lnth\ (kfilter\ p\ na\ as)\ n - na = lnth\ (kfilter\ p\ 0\ as)\ n$
using *kfilter-lnth-n-zero* **by** *blast*
have $eSuc\ (epred\ (llength\ nella)) = enat\ 0 + llength\ nella$
using *a4* **by** (*metis co.enat.exhaust-sel gen-llength-def llength-code llength-eq-0*)
then have $enat\ (the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))) <$
$llength\ (kfilter\ Pa\ na\ nella) \longrightarrow$
$enat\ (lnth\ (kfilter\ Pa\ na\ nella)$
$(the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))) - na)$
$< eSuc\ (epred\ (llength\ nella))$
using *f9 f8 f7 f6 the-enat.simps* **by** *presburger*
then have *f10*: $enat\ (the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))) <$
$llength\ (kfilter\ Pa\ na\ nella) \longrightarrow$
$enat\ (lnth\ (kfilter\ Pa\ na\ nella)$
$(the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))) - na)$
$\leq epred\ (llength\ nella)$
using *iless-Suc-eq* **by** *blast*
have *f11*: $\forall\ n\ p\ na\ as.\ \neg\ enat\ n < llength\ (kfilter\ p\ na\ as) \vee lnth\ as\ (lnth\ (kfilter\ p\ na\ as)\ n - na) =$
$(lnth\ (lfilter\ p\ as)\ n::'a)$
using *lfilter-kfilter* **by** *blast*
have *f12*: $the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (lfilter\ Pa\ nella)))) =$
$the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))$
by (*simp add*: *kfilter-llength*)
have $\neg\ lnull\ (kfilter\ Pa\ na\ nella)$
using *a1 a2 kfilter-not-lnull-conv* **by** *auto*
then have $enat\ (the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))) <$
$llength\ (kfilter\ Pa\ na\ nella)$
using *f7 a3* **by** (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 the-enat.simps*)
then show $\neg\ lnull\ nella \Longrightarrow$
$enat\ ia \leq epred\ (llength\ (kfilter\ Pa\ na\ nella)) \Longrightarrow$
$x \in lset\ nella \Longrightarrow$
$Pa\ x \Longrightarrow$
$lnth\ nella\ (the\text{-}enat\ (min\ (enat\ (lnth\ (kfilter\ Pa\ na\ nella)\ ia - na))\ (epred\ (llength\ nella)))) =$
$lnth\ (lfilter\ Pa\ nella)\ (the\text{-}enat\ (min\ (enat\ ia)\ (epred\ (llength\ (lfilter\ Pa\ nella)))))$
using *f12 f11 f10 f5* **by** *simp*

qed

lemma *in-nkfilter-nset*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $x \in nset(nkfilter\ P\ n\ nell) \longleftrightarrow x \in \{\ n{+}i \mid i.\ i \leq nlength\ nell \wedge P\ (nnth\ nell\ i)\}$
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv min-def*,
metis co.enat.exhaust-sel gen-llength-def iless-Suc-eq kfilter-holds kfilter-llength-n-zero kfilter-lnth-n-zero kfilter-lowerbound kfilter-upperbound ldistinct-Ex1 ldistinct-kfilter le-add-diff-inverse llength-code llength-eq-0,
metis add.commute co.enat.exhaust-sel iless-Suc-eq kfilter-holds-not-a llength-eq-0)

lemma *nkfilter-nset*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $nset(nkfilter\ P\ n\ nell) = \{\ n{+}i \mid i.\ i \leq nlength\ nell \wedge P\ (nnth\ nell\ i)\}$
using *assms in-nkfilter-nset*[*of nell P - n*] **by** *blast*

lemma *nlength-nkfilter-le* [*simp*]:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $nlength\ (nkfilter\ P\ n\ nell) \leq nlength\ nell$
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv epred-le-epredI kfilter-llength llength-lfilter-ile*)

lemma *nkfilter-nleast*:
assumes $\exists\ x \in nset\ xs.\ P\ x$
shows $nnth\ (nkfilter\ P\ n\ xs)\ 0 = n{+}(nleast\ P\ xs)$
using *assms*
by *transfer*
(*auto simp add*: *kfilter-not-lnull-conv kfilter-lnull-conv*,
metis enat-min-eq-0-iff kfilter-lnth-zero kfilter-lnull-conv the-enat-0 zero-enat-def)

lemma *ndistinct-nkfilter*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $ndistinct(nkfilter\ P\ n\ nell)$
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv ldistinct-kfilter*)

lemma *nkfilter-ndropn-nset*:
assumes $\exists\ x \in nset\ (ndropn\ k\ nell).\ P\ x$
$k \leq nlength\ nell$
shows $nset(nkfilter\ P\ n\ (ndropn\ k\ nell)) = \{\ n{+}i | i.\ i{\leq}nlength\ nell - k \wedge P\ (nnth\ nell\ (k{+}i))\}$
using *assms nkfilter-nset*[*of* $(ndropn\ k\ nell)\ P\ n$]
ndropn-nnth[*of - nell k*] **by** *auto*

lemma *nkfilter-ndropn-nset-b*:
assumes $\exists\ x \in nset\ (ndropn\ k\ nell).\ P\ x$
$k \leq nlength\ nell$

shows *nset(nkfilter P n (ndropn k nell)) = { n+ i |i. i≤ nlength nell − k ∧ P (nnth nell (i+k))}*
proof −
have *1*: *{ n+i|i. i≤nlength nell − k ∧ P (nnth nell (k+i))} =*
{ n+ i |i. i≤ nlength nell − k ∧ P (nnth nell (i+k))}
using *assms* **by** (*auto simp add*: *add.commute*)
show *?thesis* **using** *assms* **by** (*simp add*: *1 nkfilter-ndropn-nset*)
qed

lemma *nfilter-nkfilter-ntaken-nlength-0*:
assumes *P (nnth nell ((nnth (nkfilter P n nell) k) −n))*
k ≤ nlength (nfilter P nell)
shows *(∃ x ∈ nset nell. P x)*
using *assms*
by (*metis enat-ile in-nset-conv-nnth linorder-le-cases ntaken-all ntaken-nlast*)

lemma *nkfilter-nlength-n-zero*:
assumes *(∃ x ∈ nset nell. P x)*
shows *nlength(nkfilter P n nell) = nlength(nkfilter P 0 nell)*
using *assms* **by** (*simp add*: *nkfilter-nlength*)

lemma *nkfilter-nnth-n-zero*:
assumes *(∃ x ∈ nset nell. P x)*
k ≤ nlength(nkfilter P n nell)
shows *(nnth (nkfilter P n nell) k) − n = (nnth (nkfilter P 0 nell) k)*
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
metis kfilter-lnull-conv,
metis kfilter-lnull-conv,
metis co.enat.exhaust-sel iless-Suc-eq kfilter-llength-n-zero kfilter-lnth-n-zero llength-eq-0
min.orderE the-enat.simps)

lemma *nkfilter-n-zero*:
assumes *(∃ x ∈ nset nell. P x)*
shows *(nkfilter P n nell) = nmap (λi. i+n) (nkfilter P 0 nell)*
proof −
have *1*: *nlength(nkfilter P n nell) = nlength (nmap (λi. i+n) (nkfilter P 0 nell))*
using *assms nkfilter-nlength-n-zero* **by** *fastforce*
have *2*: *⋀ k. k ≤ nlength (nkfilter P n nell) ⟶*
nnth (nkfilter P n nell) k = nnth (nmap (λi. i+n) (nkfilter P 0 nell)) k
using *1 assms nkfilter-nnth-n-zero*[*of nell P - n*]
by (*metis diff-add nkfilter-lowerbound nlength-nmap nnth-nmap*)
show *?thesis* **by** (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

lemma *nkfilter-n-zero-a*:
assumes *(∃ x ∈ nset nell. P x)*
shows *(nkfilter P 0 nell) = nmap (λi. i−n) (nkfilter P n nell)*
proof −
have *1*: *nlength(nkfilter P 0 nell) = nlength (nmap (λi. i−n) (nkfilter P n nell))*

by (*metis assms nkfilter-nlength-n-zero nlength-nmap*)
have *2*: $\bigwedge k.\ k \leq nlength(nkfilter\ P\ 0\ nell) \longrightarrow$
$nnth\ (nkfilter\ P\ 0\ nell)\ k = nnth\ (nmap\ (\lambda i.\ i-n)\ (nkfilter\ P\ n\ nell))\ k$
by (*simp add*: *1 assms nkfilter-nnth-n-zero*)
show *?thesis* **by** (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

lemma *nkfilter-holds*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$y \in nset(nkfilter\ P\ n\ nell)$
shows $P\ (nnth\ nell\ (y-n))$
using *assms in-nkfilter-nset*[*of nell P*] **by** *force*

lemma *nkfilter-holds-not*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$y \in \{i+n\ |i.\ i \leq nlength\ nell\} - (nset\ (nkfilter\ P\ n\ nell))$
shows $\neg\ P\ (nnth\ nell\ (y-n))$
using *assms nkfilter-nset*[*of nell P n*] **by** *auto*

lemma *nkfilter-holds-a*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i \leq nlength\ nell$
$(i+n) \in nset(nkfilter\ P\ n\ nell)$
shows $P\ (nnth\ nell\ i)$
using *assms nkfilter-holds*[*of nell P i+n n*] **by** *simp*

lemma *nkfilter-holds-not-a*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i \leq nlength\ nell$
$P\ (nnth\ nell\ i)$
shows $(i+n) \in\ nset(nkfilter\ P\ n\ nell)$
using *assms* **by** (*simp add*: *in-nkfilter-nset*)

lemma *nkfilter-holds-b*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i \leq nlength\ nell$
shows $(i+n) \in nset(nkfilter\ P\ n\ nell) = P\ (nnth\ nell\ i)$
using *assms* **by** (*meson nkfilter-holds-a nkfilter-holds-not-a*)

lemma *nkfilter-holds-c*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$n \leq i$
$i-n \leq nlength\ nell$
shows $i \in nset(nkfilter\ P\ n\ nell) = P\ (nnth\ nell\ (i-n))$
using *assms*
by (*metis diff-add idiff-enat-enat nkfilter-holds-b*)

lemma *nkfilter-holds-not-b*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$n \leq i$

$i-n \leq$ *nlength nell*
shows $i \notin$ *nset* (*nkfilter P n nell*) = ($\neg$ *P* (*nnth nell* (*i*−*n*)))
using *assms*
by (*simp add*: *nkfilter-holds-c*)

lemma *nkfilter-disjoint-nset-coset*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
shows ({*i*+*n*|*i*. *i* $\leq$ *nlength nell*} − *nset*(*nkfilter P n nell*)) $\cap$ *nset* (*nkfilter P n nell*) = {}
using *assms* **by** (*simp add*: *inf.commute*)

lemma *nidx-nkfilter-expand*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
(*Suc i*) $\leq$ *nlength*(*nkfilter P n nell*)
shows *nnth* (*nkfilter P n nell*) *i* < *nnth* (*nkfilter P n nell*) (*Suc i*)
using *assms* **by** (*simp add*: *nkfilter-mono*)

lemma *nidx-nkfilter*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
shows *nidx* (*nkfilter P n nell*)
using *assms nidx-nkfilter-expand*[*of nell P - n*] *nidx-expand*[*of* (*nkfilter P n nell*)]
by *blast*

lemma *nidx-nkfilter-gr-eq*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
k $\leq$ *j*
j $\leq$ *nlength* (*nkfilter P n nell*)
shows *nnth* (*nkfilter P n nell*) *k* $\leq$ *nnth* (*nkfilter P n nell*) *j*
using *assms nidx-nkfilter*[*of nell P n*] *nidx-less-eq*[*of nkfilter P n nell k j*]
by *blast*

lemma *nidx-nkfilter-gr*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
shows ($\forall$ *j*. *k*<*j* $\wedge$ *j* $\leq$ *nlength* (*nkfilter P n nell*) $\longrightarrow$
nnth (*nkfilter P n nell*) *k* < *nnth*(*nkfilter P n nell*) *j*)
using *assms nidx-nkfilter*[*of nell P n*] *nidx-less*[*of nkfilter P n nell*]
using *less-imp-Suc-add* **by** *blast*

lemma *nidx-nkfilter-less-eq*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
k $\leq$ *nlength* (*nkfilter P n nell*)
shows $\forall$*j*. *j* $\leq$ *k* $\longrightarrow$ *nnth* (*nkfilter P n nell*) *j* $\leq$ *nnth* (*nkfilter P n nell*) *k*
using *assms* **by** (*simp add*: *nidx-nkfilter-gr-eq*)

lemma *nkfilter-not-before*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
i < (*nnth* (*nkfilter P 0 nell*) *0*)
shows $\neg$ *P* (*nnth nell i*)
using *assms*
by *transfer*

(*auto simp add*: *min-def split*: *if-split-asm*,
meson kfilter-not-before llength-eq-0 not-gr-zero,
metis dual-order.strict-trans kfilter-not-before less-enatE llength-eq-0 not-gr-zero
not-le-imp-less the-enat.simps,
meson le-less-linear less-enatE less-nat-zero-code,
meson le-less-linear less-enatE not-less0)

lemma *nkfilter-n-not-before*:
assumes ($\exists$ $x \in$ *nset* (*ndropn n nell*). *P x*)
$n \leq$ *nlength nell*
$n \leq i$
$i <$ (*nnth* (*nkfilter P n* (*ndropn n nell*)) *0*)
shows $\neg$ *P* (*nnth nell i*)
using *assms*
apply *transfer*
unfolding *min-def*
using *zero-enat-def*
proof (*auto split*: *if-split-asm*)
show $\bigwedge$*n nell P i x*.
enat 0 = 0 $\Longrightarrow$
$\neg$ *lnull nell* $\Longrightarrow$
enat n $\leq$ *epred* (*llength nell*) $\Longrightarrow$
$n \leq i \Longrightarrow$
$\neg$ *lnull* (*kfilter P n* (*ldropn n nell*)) $\Longrightarrow$
$i <$ *lnth* (*kfilter P n* (*ldropn n nell*)) *0* $\Longrightarrow$
x $\in$ *lset* (*ldropn n nell*) $\Longrightarrow$ *P x* $\Longrightarrow$ *enat i* $\leq$ *epred* (*llength nell*) $\Longrightarrow$ *P* (*lnth nell i*) $\Longrightarrow$ *False*
by (*metis co.enat.exhaust-sel iless-Suc-eq kfilter-n-not-before llength-eq-0 not-gr-zero*)
next
fix *na* :: *nat* **and** *nella* :: *'b llist* **and** *Pa* :: *'b* $\Rightarrow$ *bool* **and** *ia* :: *nat* **and** *x* :: *'b*
assume *a1*: *ia <* *lnth* (*kfilter Pa na* (*ldropn na nella*)) *0*
assume *a2*: $\neg$ *lnull* (*kfilter Pa na* (*ldropn na nella*))
assume *a3*: *enat na* $\leq$ *epred* (*llength nella*)
assume *a4*: $\neg$ *lnull nella*
assume *a5*: $\neg$ *enat ia* $\leq$ *epred* (*llength nella*)
assume *a6*: *Pa* (*lnth nella* (*the-enat* (*epred* (*llength nella*))))
have *f7*: $\forall$ *p n bs*. *lnull* (*kfilter p n* (*bs*::*'b llist*)) $\vee$ *lnth* (*kfilter p n bs*) *0* = *n* + *lleast p bs*
by (*meson kfilter-lnth-zero*)
then have *f8*: *lnth* (*kfilter Pa na* (*ldropn na nella*)) *0* = *na* + *lleast Pa* (*ldropn na nella*)
using *a2* **by** *blast*
have *f9*: *0 <* *llength* (*kfilter Pa na* (*ldropn na nella*))
using *a2* **by** *simp*
have *f10*: *na* $\leq$ *the-enat* (*epred* (*llength nella*))
by (*metis a3 a5 enat-ord-code*(*4*) *enat-ord-simps*(*1*) *enat-the-enat less-imp-le*)
have *f11*: *the-enat* (*epred* (*llength nella*)) < *lnth* (*kfilter Pa na* (*ldropn na nella*)) *0*
by (*metis a1 a5 dual-order.strict-trans enat-ord-simps*(*2*) *enat-ord-simps*(*3*) *enat-the-enat*
not-le-imp-less)
then show *False* **using** *kfilter-n-not-before*[*of Pa na nella* (*the-enat* (*epred* (*llength nella*)))]
using *f10 f11*
by (*metis a3 a4 a6 co.enat.exhaust-sel f9 iless-Suc-eq llength-eq-0*)
qed

lemma *nkfilter-not-after*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$nnth\ (nkfilter\ P\ 0\ nell)\ k < i$
$nlength(nkfilter\ P\ 0\ nell) = (enat\ k)$
$i \leq nlength\ nell$
shows $\neg\ P\ (nnth\ nell\ i)$
using *assms*
by *transfer*
(*auto simp add*: *min-def split*: *if-split-asm*,
metis co.enat.exhaust-sel diff-Suc-1 eSuc-enat iless-Suc-eq kfilter-not-after llength-eq-0
not-gr-zero)

lemma *nkfilter-n-not-after*:
assumes $(\exists\ x \in nset\ (ndropn\ n\ nell).\ P\ x)$
$n \leq nlength\ nell$
$nnth\ (nkfilter\ P\ n\ (ndropn\ n\ nell))\ k < i$
$nlength(nkfilter\ P\ n\ (ndropn\ n\ nell)) = (enat\ k)$
$i \leq nlength\ nell$
shows $\neg\ P\ (nnth\ nell\ i)$
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
metis diff-Suc-1 eSuc-enat eSuc-epred iless-Suc-eq kfilter-n-not-after llength-eq-0 not-gr-zero)

lemma *nkfilter-not-between*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$nnth\ (nkfilter\ P\ 0\ nell)\ k < i$
$i < nnth\ (nkfilter\ P\ 0\ nell)\ (Suc\ k)$
$(Suc\ k) \leq nlength\ (nkfilter\ P\ 0\ nell)$
shows $\neg\ P\ (nnth\ nell\ i)$
using *assms*
by *transfer*
(*auto simp add*: *min-def split*: *if-split-asm*,
metis co.enat.exhaust-sel iless-Suc-eq kfilter-not-between llength-eq-0,
metis co.enat.exhaust-sel gen-llength-def iless-Suc-eq kfilter-upperbound le-less-trans
llength-code llength-eq-0 min.cobounded2 min.strict-order-iff min-enat-simps(1),
meson Suc-ile-eq less-imp-le,
meson Suc-ile-eq dual-order.strict-implies-order)

lemma *ntaken-nset*:
assumes $k \leq nlength\ nell$
shows $nset\ (ntaken\ k\ nell) = \{\ (nnth\ nell\ i)\ |\ i.\ i \leq k\}$
using *assms*
by (*auto simp add*: *in-nset-conv-nnth*)
(*metis min.orderE ntaken-nnth*,
metis min.orderE ntaken-nnth)

lemma *ndropn-nset*:
assumes $k \leq$ *nlength nell*
shows *nset* (*ndropn k nell*) = { (*nnth nell i*) | *i*. $k \leq i \wedge i \leq$ *nlength nell*}
using *assms*
by (*auto simp add*: *in-nset-conv-nnth*)
(*metis add.commute enat-min-eq le-add1 plus-enat-simps*(*1*),
metis enat-minus-mono1 idiff-enat-enat le-add-diff-inverse)

lemma *nfilter-nkfilter-ntaken-nidx-a*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*ntaken k* (*nkfilter P n nell*))
using *assms*
by (*simp add*: *nidx-expand nidx-nkfilter-gr ntaken-nnth*)

lemma *nfilter-nkfilter-ndropn-nidx-a*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*ndropn k* (*nkfilter P n nell*))
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv*,
metis (*no-types*, *lifting*) *gr-implies-not-zero kfilter-llength leI lfilter-kfilter-ldropn-lidx-a*
lidx-def llength-eq-0 lnull-ldropn)

lemma *nfilter-nkfilter-ntaken-nidx-b*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms nidx-nkfilter*[*of* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *P 0*]
by (*metis nfinite-ntaken nset-nlast ntaken-nlast*)

lemma *nfilter-nkfilter-ntaken-nidx-b-1*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms nfilter-nkfilter-ntaken-nidx-b*[*of P nell k*] *nkfilter-holds*[*of nell P*]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *nfilter-nkfilter-ntaken-nidx-b-2*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P n nell*) *k*) $-n$))
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P n* (*ntaken* (*nnth* (*nkfilter P n nell*) *k*) *nell*))
using *assms nidx-nkfilter*[*of* (*ntaken* (*nnth* (*nkfilter P n nell*) *k*) *nell*) *P n*]
by (*metis* (*mono-tags*, *lifting*) *diff-le-self linorder-le-cases mem-Collect-eq*
nfilter-nkfilter-ntaken-nlength-0 ntaken-all ntaken-nset)

lemma *nfilter-nkfilter-ntaken-nidx-b-3*:

assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nidx\ (nkfilter\ P\ n\ (ntaken\ (nnth\ (nkfilter\ P\ n\ nell)\ k)\ nell))$
using *assms nfilter-nkfilter-ntaken-nidx-b-2*[*of P nell n k*] *nkfilter-holds*
by (*metis in-nset-conv-nnth nkfilter-nlength*)

lemma *nfilter-nkfilter-ndropn-nidx-b*:
assumes $P\ (nnth\ nell\ (\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)))$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nidx\ (nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
using *assms nidx-nkfilter*[*of* $(ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)\ P\ 0$]
by (*metis diff-add diff-zero in-nset-conv-nnth ndropn-nnth zero-enat-def zero-le*)

lemma *nfilter-nkfilter-ndropn-nidx-b-1*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nidx\ (nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
using *assms nfilter-nkfilter-ndropn-nidx-b*[*of P nell k*] *nkfilter-holds*[*of nell P*]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *nfilter-nkfilter-ndropn-nidx-b-2*:
assumes $P\ (nnth\ nell\ (\ (nnth\ (nkfilter\ P\ n\ nell)\ k)\ -n))$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nidx\ (nkfilter\ P\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n)$
$(ndropn\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n)\ nell)\ )$
proof −
have *1*: $enat\ (nnth\ (nkfilter\ P\ n\ nell)\ k)\ -n \leq nlength\ nell$
using *nkfilter-upperbound*[*of nell P*] *nkfilter-nnth-n-zero*[*of nell P k n*] *assms*
by (*metis gen-nlength-def idiff-enat-enat nfilter-nkfilter-ntaken-nlength-0*
nkfilter-nlength nlength-code)
have *2*: $nset\ (ndropn\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n)\ nell) =$
$\{\ (nnth\ nell\ i)\ |i.\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n) \leq i \wedge i \leq nlength\ nell\ \}$
using *ndropn-nset*[*of* $(nnth\ (nkfilter\ P\ n\ nell)\ k){-}n\ nell$] *1* **by** *simp*
have *3*: $\exists x{\in}nset\ (ndropn\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n)\ nell).\ P\ x$
using *1 2 assms*(*1*) **by** *auto*
show *?thesis* **using** *3*
nidx-nkfilter[*of* $(ndropn\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n)\ nell)\ P\ (nnth\ (nkfilter\ P\ n\ nell)\ k){-}n$]
by *blast*
qed

lemma *nfilter-nkfilter-ndropn-nidx-b-3*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nidx\ (nkfilter\ P\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n)$
$(ndropn\ ((nnth\ (nkfilter\ P\ n\ nell)\ k){-}n)\ nell))$
using *assms nfilter-nkfilter-ndropn-nidx-b-2*
by (*metis in-nset-conv-nnth nkfilter-holds nkfilter-nlength*)

lemma *ntaken-nkfilter-ntaken-nset-eq*:
assumes $P\ (nnth\ nell\ (\ (nnth\ (nkfilter\ P\ 0\ nell)\ k))\ )$

```
      k ≤ nlength (nfilter P nell)
 shows    nset(ntaken k (nkfilter P 0 nell)) =
        nset(nkfilter P 0 (ntaken ((nnth (nkfilter P 0 nell) k)) nell))
proof −
 have 1: ( (nnth (nkfilter P 0 nell) k)) ≤ nlength nell
  using assms nkfilter-upperbound[of nell P k 0]
  by (metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code)
 have 2: ∃ x ∈ nset nell. P x
  using assms by (metis 1 exists-Pred-nnth-nset)
 have 3: {i. i ≤ nlength(ntaken (nnth (nkfilter P 0 nell) k) nell) ∧
          P (nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) i) }
        = {i. i ≤ (nnth (nkfilter P 0 nell) k) ∧ P (nnth nell i)}
  by (auto simp add: ntaken-nnth 1 order-subst2)
 have 4: ∃ x ∈ nset(ntaken (nnth (nkfilter P 0 nell) k) nell). P x
  using 1 assms(1) ntaken-nset by fastforce
 have 5: {i. i ≤ (nnth (nkfilter P 0 nell) k) ∧ P (nnth nell i)} =
       {i. i ≤ (nnth (nkfilter P 0 nell) k) ∧ i ∈ nset(nkfilter P 0 nell)}
  using 4 1 2 nkfilter-holds-b
  by (metis (mono-tags, opaque-lifting) add-cancel-left-right enat-ord-simps(1) order.trans)
 have 6: {i. i ≤ (nnth (nkfilter P 0 nell) k) ∧ i ∈ nset(nkfilter P 0 nell)} =
       {i. i≤ (nnth (nkfilter P 0 nell) k) ∧
         i∈ { (nnth (nkfilter P 0 nell) j) | j. j ≤ nlength (nkfilter P 0 nell)}}
  by (simp add: nset-conv-nnth)
 have 7: {i. i≤ (nnth (nkfilter P 0 nell) k) ∧
          i∈ { (nnth (nkfilter P 0 nell) j) | j. j ≤ nlength (nkfilter P 0 nell)}} =
        {(nnth (nkfilter P 0 nell) j) |j. j ≤ k}
  by (auto simp add: assms 2 nidx-nkfilter-less-eq nkfilter-nlength)
     (metis 2 dual-order.antisym le-cases nidx-nkfilter-gr-eq nkfilter-nlength,
     metis assms(2) dual-order.trans enat-ord-simps(1))
 have 8: k ≤ nlength (nkfilter P 0 nell)
  by (simp add: 2 assms(2) nkfilter-nlength)
 have 9: {(nnth (nkfilter P 0 nell) j) |j. j ≤ k} = nset(ntaken k (nkfilter P 0 nell))
  using ntaken-nset[of k (nkfilter P 0 nell)] 8 by auto
 have 91: min (enat (nnth (nkfilter P 0 nell) k)) (nlength nell) = (nnth (nkfilter P 0 nell) k)
  by (simp add: 1 min-def)
 have 10: nset(nkfilter P 0 (ntaken ((nnth (nkfilter P 0 nell) k)) nell)) =
          {i. i ≤ (nnth (nkfilter P 0 nell) k) ∧
              P (nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) i) }
  using nkfilter-nset[of (ntaken (nnth (nkfilter P 0 nell) k) nell) P 0 ]
  1 4 91 ntaken-nlength[of (nnth (nkfilter P 0 nell) k) nell]
  by auto
 have 11: nlength((ntaken ((nnth (nkfilter P 0 nell) k)) nell)) = (nnth (nkfilter P 0 nell) k)
  by (simp add: 1)
 have 12: {i. i ≤ (nnth (nkfilter P 0 nell) k) ∧
              P (nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) i) } =
          {i. i ≤ nlength((ntaken ((nnth (nkfilter P 0 nell) k)) nell)) ∧
              P (nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) i) }
  using 11 by auto
show ?thesis
using 10 12 3 5 6 7 9 by auto
```

qed

lemma *ntaken-nkfilter-ntaken-nset-eq-1*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nset(ntaken\ k\ (nkfilter\ P\ 0\ nell)) =$
$nset(nkfilter\ P\ 0\ (ntaken\ ((nnth\ (nkfilter\ P\ 0\ nell)\ k))\ nell))$
using *assms ntaken-nkfilter-ntaken-nset-eq*[*of P nell k*]
nkfilter-holds[*of nell P* (*nnth* (*nkfilter P 0 nell*) *k*) *0*]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *ndropn-nkfilter-ndropn-nset-eq*:
assumes $P\ (nnth\ nell\ (\ (nnth\ (nkfilter\ P\ 0\ nell)\ k))\ )$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nset(ndropn\ k\ (nkfilter\ P\ 0\ nell)) =$
$nset(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ (ndropn\ ((nnth\ (nkfilter\ P\ 0\ nell)\ k))\ nell))$
proof −
have *1*: $(nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq nlength\ nell$
using *nkfilter-upperbound*[*of nell P k 0*] *assms*
by (*metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code*)
have *2*: $\exists\ x \in nset\ nell.\ P\ x$
using *assms* **by** (*metis 1 exists-Pred-nnth-nset*)
have *4*: $\exists\ x \in nset(ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell).\ P\ x$
using *1 assms*(*1*) *ndropn-nset* **by** *fastforce*
have *10*: $nset(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)$
$(ndropn\ ((nnth\ (nkfilter\ P\ 0\ nell)\ k))\ nell)) =$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i |\ i.\ i \leq nlength\ nell - (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$P\ (nnth\ nell\ (i+\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)))\ \}$
using *1*
nkfilter-ndropn-nset-b[*of* (*nnth* (*nkfilter P 0 nell*) *k*) *nell P* (*nnth* (*nkfilter P 0 nell*) *k*)]
4 **by** *linarith*
have *5*: $\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i |\ i.\ i \leq nlength\ nell - (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$P\ (nnth\ nell\ (i+\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)))\ \} =$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i |\ i.\ i \leq nlength\ nell - (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$(i+\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)) \in nset(nkfilter\ P\ 0\ nell)\}$
proof (*auto simp add*: *add.commute*)
fix *i* :: *nat*
assume *a1*: *enat i* ≤ *nlength nell* − *enat* (*nnth* (*nkfilter P 0 nell*) *k*)
assume *a2*: *P* (*nnth nell* (*i* + *nnth* (*nkfilter P 0 nell*) *k*))
have *f0*: *enat* (*i* + (*nnth* (*nkfilter P 0 nell*) *k*)) ≤ *nlength nell*
using *1 a1 enat-min-eq* **by** *auto*
show *i* + *nnth* (*nkfilter P 0 nell*) *k* ∈ *nset* (*nkfilter P 0 nell*)
by (*metis Nat.add-0-right a2 f0 in-nset-conv-nnth nkfilter-holds-b*)
next
fix *i*
assume *b0*: *enat i* ≤ *nlength nell* − *enat* (*nnth* (*nkfilter P 0 nell*) *k*)
assume *b1*: *i* + *nnth* (*nkfilter P 0 nell*) *k* ∈ *nset* (*nkfilter P 0 nell*)
show *P* (*nnth nell* (*i* + *nnth* (*nkfilter P 0 nell*) *k*))
using *nkfilter-holds*[*of nell P*] *2* **by** (*metis b1 minus-nat.diff-0*)
qed

have *51*: $\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i |\ i.\ i \leq nlength\ nell - (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$(i+ (nnth\ (nkfilter\ P\ 0\ nell)\ k)) \in nset(nkfilter\ P\ 0\ nell)\} =$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i |\ i.$
$(nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq nlength\ nell\ \wedge$
$(i+ (nnth\ (nkfilter\ P\ 0\ nell)\ k)) \in nset(nkfilter\ P\ 0\ nell)\}$
by *auto*
(*metis 2 add.left-neutral in-nset-conv-nnth nkfilter-upperbound zero-enat-def*,
metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat)
have *52*: $\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i |\ i.$
$(nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq nlength\ nell\ \wedge$
$(i+ (nnth\ (nkfilter\ P\ 0\ nell)\ k)) \in nset(nkfilter\ P\ 0\ nell)\} =$
$\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in nset(nkfilter\ P\ 0\ nell)\}$
by (*metis* (*no-types*, *lifting*) *add.commute diff-add*)
have *53* : $\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in nset(nkfilter\ P\ 0\ nell)\} =$
$\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ jj)\ |jj.\ jj \leq nlength\ (nkfilter\ P\ 0\ nell)\}\}$
by (*simp add*: *nset-conv-nnth*)
have *54*: $\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ jj)\ |jj.\ jj \leq nlength\ (nkfilter\ P\ 0\ nell)\}\} =$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ j)\ |j.\ k \leq j \wedge j \leq nlength(nkfilter\ P\ 0\ nell)\}$
using *assms 2*
by (*auto*,
metis dual-order.antisym le-cases nidx-less-eq nidx-nkfilter nkfilter-nlength,
meson nidx-nkfilter-gr-eq,
metis gen-nlength-def nkfilter-upperbound nlength-code)
have *8*: $k \leq nlength\ (nkfilter\ P\ 0\ nell)$
by (*simp add*: *2 assms*(*2*) *nkfilter-nlength*)
have *9*: $\{(nnth\ (nkfilter\ P\ 0\ nell)\ j)\ |j.\ k \leq j \wedge j \leq nlength(nkfilter\ P\ 0\ nell)\} =$
$nset(ndropn\ k\ (nkfilter\ P\ 0\ nell))$
by (*simp add*: *8 ndropn-nset*)
show *?thesis*
using *10 5 51 52 53 54 9* **by** *auto*
qed

lemma *ndropn-nkfilter-ndropn-nset-eq-1*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $nset(ndropn\ k\ (nkfilter\ P\ 0\ nell)) =$
$nset(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ (ndropn\ ((nnth\ (nkfilter\ P\ 0\ nell)\ k))\ nell))$
using *assms ndropn-nkfilter-ndropn-nset-eq*[*of P nell k*]
by (*metis diff-zero in-nset-conv-nnth nkfilter-holds nkfilter-nlength*)

lemma *nkfilter-nkfilter-ntaken*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $ntaken\ k\ (nkfilter\ P\ 0\ nell) =$
$(nkfilter\ P\ 0\ (ntaken\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
using *assms*

by (*simp add*: *nfilter-nkfilter-ntaken-nidx-a nfilter-nkfilter-ntaken-nidx-b-1 nidx-nset-eq ntaken-nkfilter-ntaken-nset-eq-1*)

lemma *nkfilter-nkfilter-ndropn*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows *ndropn k* (*nkfilter P 0 nell*) =
(*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
proof −
have *1*: *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
using *nkfilter-holds*[*of nell P*] *assms*
by (*metis diff-zero in-nset-conv-nnth nkfilter-nlength*)
show *?thesis*
by (*metis 1 assms*(*1*) *assms*(*2*) *diff-zero ndropn-nkfilter-ndropn-nset-eq nfilter-nkfilter-ndropn-nidx-a nfilter-nkfilter-ndropn-nidx-b-3 nidx-nset-eq*)
qed

lemma *nkfilter-nmap-nfilter*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows *nmap* (λ*n*. *nnth nell n*) (*nkfilter P 0 nell*) = *nfilter P nell*
using *assms nellist-eq-nnth-eq*[*of nmap* (λ*n*. *nnth nell n*) (*nkfilter P 0 nell*) *nfilter P nell*]
nkfilter-nfilter[*of nell P - 0*] *nkfilter-nnth-n-zero*[*of nell P - 0*]
by (*simp add*: *nkfilter-nlength*)

lemma *nfilter-nkfilter-ntaken*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
$k \leq nlength\ (nkfilter\ P\ 0\ nell)$
shows *ntaken k* (*nfilter P nell*) =
nfilter P (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)
proof −
have *1*: $\exists\ x \in nset\ nell.\ P\ x$
using *assms*
by (*metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *2*: *nfilter P nell* = *nmap* (λ*n*. *nnth nell n*) (*nkfilter P 0 nell*)
using *1 assms* **by** (*simp add*: *nkfilter-nmap-nfilter*)
have *3*: *ntaken k* (*nfilter P nell*) = *ntaken k* (*nmap* (λ*n*. *nnth nell n*) (*nkfilter P 0 nell*))
using *2* **by** *simp*
have *4*: *ntaken k* (*nmap* (λ*n*. *nnth nell n*) (*nkfilter P 0 nell*)) =
nmap (λ*n*. *nnth nell n*) (*ntaken k* (*nkfilter P 0 nell*))
by *simp*
have *5*: $\exists\ x \in nset\ (ntaken\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell).\ P\ x$
using *assms* **by** (*metis nfinite-ntaken nset-nlast ntaken-nlast*)
have *6*: (*nfilter P* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap (λ*s*. *nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *s*)
(*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *5* **by** (*simp add*: *nkfilter-nmap-nfilter*)
have *7*: *nmap* (λ*n*. *nnth nell n*) (*ntaken k* (*nkfilter P 0 nell*)) =

```
      nmap (λn. nnth nell n) (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
  by (simp add: 1 2 assms(2) nkfilter-nkfilter-ntaken)
 have 70: enat k ≤ nlength (nfilter P nell)
  using 2 assms by auto
 have 71: (⋀z. z ∈ nset (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell)) ⟹
     (λn. nnth nell n) z = (λs. nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) s)  z)
  using assms 1 70 ntaken-nkfilter-ntaken-nset-eq[of P nell k]
     ntaken-nset[of k (nkfilter P 0 nell)] mem-Collect-eq
    nidx-nkfilter-gr-eq[of nell P - - 0]
  proof simp
    fix z :: nat
    assume a1: enat k ≤ nlength (nkfilter P 0 nell)
    assume a2: ⋀k j. ⟦k ≤ j; enat j ≤ nlength (nkfilter P 0 nell)⟧ ⟹
                  nnth (nkfilter P 0 nell) k ≤ nnth (nkfilter P 0 nell) j
    assume a3: z ∈ nset (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
    assume {nnth (nkfilter P 0 nell) i |i. i ≤ k} =
          nset (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
    then have ∃n. z = nnth (nkfilter P 0 nell) n ∧ n ≤ k
    using a3 by blast
    then have z ≤ nnth (nkfilter P 0 nell) k
    using a2 a1 by meson
    then show nnth nell z = nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) z
    by (simp add: ntaken-nnth)
  qed
 have 8: nmap (λn. nnth nell n)
            (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell)) =
        nmap (λs. nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) s)
            (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
  using 1 5 assms nellist.map-cong0
  using 71 by blast
 show ?thesis
 by (simp add: 3 6 7 8)
qed

lemma nfilter-nkfilter-ntaken-1:
 assumes ∃ x ∈ nset nell. P x
       k ≤ nlength (nkfilter P 0 nell)
 shows    ntaken k (nfilter P nell) =
         nfilter P (ntaken (nnth (nkfilter P 0 nell) k) nell)
using assms nfilter-nkfilter-ntaken[of P nell k]
by (metis in-nset-conv-nnth nkfilter-holds nkfilter-nnth-n-zero)

lemma nkfilter-nmap-shift:
 assumes ∃ x ∈ nset nell. P x
 shows  nmap (λs. nnth nell (s+n)) (nkfilter P 0 nell) =
       nmap (λs. nnth nell s) (nkfilter P n nell)
proof −
 have 1: nlength (nmap (λs. nnth nell (s+n)) (nkfilter P 0 nell)) =
        nlength (nmap (λs. nnth nell s) (nkfilter P n nell))
  by (simp add: assms nkfilter-nlength)
```

have *2*: $\bigwedge i.\ i \leq$ *nlength* (*nmap* ($\lambda s.$ *nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*)) $\longrightarrow$
nnth (*nmap* ($\lambda s.$ *nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*)) *i* =
nnth nell ((*nnth* (*nkfilter P 0 nell*) *i*)+*n*)
by *simp*
have *3*: $\bigwedge i.\ i \leq$ *nlength* (*nmap* ($\lambda s.$ *nnth nell s*) (*nkfilter P n nell*)) $\longrightarrow$
nnth (*nmap* ($\lambda s.$ *nnth nell s*) (*nkfilter P n nell*)) *i* =
nnth nell (*nnth* (*nkfilter P n nell*) *i*)
by *simp*
have *4*: $\bigwedge i.\ i \leq$ *nlength* (*nmap* ($\lambda s.$ *nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*)) $\longrightarrow$
nnth nell ((*nnth* (*nkfilter P 0 nell*) *i*)+*n*) = *nnth nell* (*nnth* (*nkfilter P n nell*) *i*)
using *nkfilter-n-zero*[*of nell P n*]
by (*simp add*: *assms*)
show *?thesis* **using** *1 4 nellist-eq-nnth-eq* **by** *force*
qed

lemma *nkfilter-nmap-shift-ndropn*:
assumes $\exists\ x \in$ *nset* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*). *P x*
$k \leq$ *nlength* (*nkfilter P 0 nell*)
shows *nmap* ($\lambda s.$ *nnth nell* (*s*+(*nnth* (*nkfilter P 0 nell*) *k*)))
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap ($\lambda s.$ *nnth nell s*) (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*)
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms*
nellist-eq-nnth-eq[*of nmap* ($\lambda s.$ *nnth nell* (*s*+(*nnth* (*nkfilter P 0 nell*) *k*)))
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
nmap ($\lambda s.$ *nnth nell s*) (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*)
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))]
using *nkfilter-n-zero*[*of* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *P*
(*nnth* (*nkfilter P 0 nell*) *k*)]
by *simp*

lemma *nfilter-nkfilter-ndropn*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
$k \leq$ *nlength* (*nkfilter P 0 nell*)
shows *ndropn k* (*nfilter P nell*) =
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)
proof −
have *1*: $\exists x \in$ *nset nell*. *P x*
using *assms*
by (*metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *2*: (*nfilter P nell*) = *nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*)
by (*simp add*: *1 nkfilter-nmap-nfilter*)
have *3*: *ndropn k* (*nfilter P nell*) = *ndropn k* (*nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*))
by (*simp add*: *2*)
have *4*: *ndropn k* (*nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*)) =
nmap ($\lambda n.$ *nnth nell n*) (*ndropn k* (*nkfilter P 0 nell*))
using *ndropn-nmap* **by** *blast*
have *5*: $\exists\ x \in$ *nset*(*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*). *P x*
by (*metis add.commute add.left-neutral assms*(*1*) *in-nset-conv-nnth ndropn-nnth zero-enat-def*
zero-le)

have *6*: *nfilter P (ndropn (nnth (nkfilter P 0 nell) k) nell) =*
nmap (λs. nnth (ndropn (nnth (nkfilter P 0 nell) k) nell) s)
(nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) k) nell))
by (*metis 5 nkfilter-nmap-nfilter*)
have *7*: *nmap (λs. nnth (ndropn (nnth (nkfilter P 0 nell) k) nell) s)*
(nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) k) nell)) =
nmap (λs. nnth nell (s+(nnth (nkfilter P 0 nell) k)))
(nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) k) nell))
by (*simp add*: *add.commute*)
have *8*: *nmap (λs. nnth nell (s+(nnth (nkfilter P 0 nell) k)))*
(nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) k) nell)) =
nmap (λs. nnth nell s)
(nkfilter P (nnth (nkfilter P 0 nell) k) (ndropn (nnth (nkfilter P 0 nell) k) nell))
using *5 assms(2) nkfilter-nmap-shift-ndropn* **by** *fastforce*
have *9*: *nmap (λs. nnth nell s)*
(nkfilter P (nnth (nkfilter P 0 nell) k) (ndropn (nnth (nkfilter P 0 nell) k) nell)) =
nmap (λs. nnth nell s)
(ndropn k (nkfilter P 0 nell))
by (*simp add*: *1 2 assms(2) nkfilter-nkfilter-ndropn*)
show *?thesis* **using** *3 4 6 7 8 9* **by** *auto*
qed

lemma *nfilter-nkfilter-ndropn-1*:
assumes ∃ *x* ∈ *nset nell*. *P x*
k ≤ *nlength (nkfilter P 0 nell)*
shows *ndropn k (nfilter P nell) =*
nfilter P (ndropn (nnth (nkfilter P 0 nell) k) nell)
using *assms nfilter-nkfilter-ndropn*
by (*metis in-nset-conv-nnth minus-nat.diff-0 nkfilter-holds*)

lemma *nfilter-nkfilter-nsubn*:
assumes *P (nnth nell ((nnth (nkfilter P 0 nell) i)))*
enat i ≤ *nlength (nkfilter P 0 nell)*
P (nnth nell ((nnth (nkfilter P 0 nell) j)))
enat j ≤ *nlength (nkfilter P 0 nell)*
i ≤ *j*
shows *nsubn (nfilter P nell) i j =*
(nfilter P (nsubn nell
(nnth (nkfilter P 0 nell) i)
(nnth (nkfilter P 0 nell) j)
)
)

proof −
have *0*: *nsubn (nfilter P nell) i j = ntaken (j − i) (ndropn i (nfilter P nell))*
using *nsubn-def1* **by** *blast*
have *1*: *ntaken (j − i) (ndropn i (nfilter P nell)) =*
ntaken (j − i) (nfilter P (ndropn (nnth (nkfilter P 0 nell) i) nell))

by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nkfilter-ndropn*)
have *4*: ((*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*)) +
(*nnth* (*nkfilter P 0 nell*) *i*)) =
((*nnth* (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *i*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*)))
proof −
have *f1*: ⋀*n ns*. (*nfirst* (*ndropn n ns*)::*nat*) = *nlast* (*ntaken n ns*)
by (*metis* (*lifting*) *ndropn-0 ndropn-nfirst ndropn-nnth ntaken-ndropn-nlast*)
have ⋀*n f ns*. *nlast* (*ntaken n* (*nmap f ns*)) = (*f* (*nlast* (*ntaken n ns*)::*nat*)::*nat*)
by *simp*
then show *?thesis*
using *f1* **by** (*metis assms*(*1*) *in-nset-conv-nnth ndropn-0 ndropn-nfirst nkfilter-n-zero zero-enat-def zero-le*)
qed
have *5*: ((*nnth* (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *i*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*))) =
(*nnth* (*nkfilter P 0 nell*) *j*)
using *assms nkfilter-nkfilter-ndropn*[*of nell P i*]
by (*metis in-nset-conv-nnth le-add-diff-inverse linorder-le-cases linorder-not-less ndropn-nnth nfinite-ntaken nkfilter-nlength nnth-beyond nset-nlast ntaken-all*)
have *10*: ∃ *x*∈*nset* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*). *P x*
by (*metis assms*(*1*) *in-nset-conv-nnth le-zero-eq nle-le nnth-zero-ndropn zero-enat-def*)
have *11*: *ndropn i* (*nfilter P nell*) = *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nkfilter-ndropn*)
have *12*: *nlength* (*ndropn i* (*nfilter P nell*)) = *nlength* (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*))
by (*simp add*: *11*)
have *13*: *nlength* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) =
nlength (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*))
by (*simp add*: *10 nkfilter-nlength*)
have *14*: *enat i* ≤ *nlength* (*nfilter P nell*)
by (*metis assms*(*1*) *assms*(*2*) *in-nset-conv-nnth linorder-le-cases nfinite-ntaken nkfilter-nlength nset-nlast*
ntaken-all ntaken-nlast)
have *15*: *enat j* ≤ *nlength* (*nfilter P nell*)
by (*metis 14 assms*(*1*) *assms*(*4*) *diff-zero nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength*)
have *16*: *enat* (*j* − *i*) ≤ *nlength* (*ndropn i* (*nfilter P nell*))
by (*metis 15 enat-minus-mono1 idiff-enat-enat ndropn-nlength*)
have *2*: *ntaken* (*j* − *i*) (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) =
(*nfilter P* (*ntaken* (*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*))
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)))
by (*metis 10 12 13 16 nfilter-nkfilter-ntaken-1*)
have *3*: (*nfilter P* (*ntaken* (*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*))
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*))) =
(*nfilter P* (*nsubn nell*
(*nnth* (*nkfilter P 0 nell*) *i*)
((*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*)) +
(*nnth* (*nkfilter P 0 nell*) *i*))
))
by (*simp add*: *ntaken-ndropn*)
show *?thesis* **using** *0 1 2 3 4 5* **by** *auto*
qed

lemma *nfilter-nkfilter-nsubn-zero*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *j*)))
enat j ≤ *nlength* (*nkfilter P 0 nell*)
shows *nsubn* (*nfilter P nell*) *0 j* =
(*nfilter P* (*nsubn nell*
0
(*nnth* (*nkfilter P 0 nell*) *j*)
)
)

by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nkfilter-ntaken nsubn-def1*)

lemma *nkfilter-nnth-aa*:
assumes ∃ *x* ∈ *nset nell*. *P x*
n ≤ *nlength* (*nfilter P nell*)
shows *P* (*nnth* (*nfilter P nell*) *n*)
using *assms in-nset-conv-nnth nkfilter-holds*[*of nell P - 0*] *nkfilter-nfilter*[*of nell P - 0*]
by (*metis nkfilter-nlength*)

lemma *nfilter-nlength-zero-conv-a*:
assumes ∃ *x* ∈ *nset nell*. *P x*
nlength(*nfilter P nell*) = *0*
shows (∃ *k* ≤ *nlength nell*. *P* (*nnth nell k*) ∧
(∀ *j*≤ *nlength nell*. *j* ≠*k* ⟶ ¬ *P* (*nnth nell j*)))
using *assms*
apply *transfer*
proof (*auto simp add*: *epred-llength min-def split*: *if-split-asm*)
fix *nella* :: *'a llist* **and** *Pa* :: *'a* ⇒ *bool* **and** *x* :: *'a*
assume *a1*: ¬ *lnull nella*
assume *a2*: *Pa x*
assume *a3*: *x* ∈ *lset nella*
assume *a4*: *lnull* (*ltl* (*lfilter Pa nella*))
show ∃ *k*. (*enat k* ≤ *llength* (*ltl nella*) ⟶
Pa (*lnth nella k*) ∧ (∀ *j*. *enat j* ≤ *llength* (*ltl nella*) ⟶ *j* ≠ *k* ⟶ ¬ *Pa* (*lnth nella j*))) ∧
enat k ≤ *llength* (*ltl nella*)
proof −
have *1*: *LCons* (*lhd nella*) (*ltl nella*) = *nella*
by (*metis a2 a3 lfilter-LNil llist.disc*(*1*) *llist.exhaust-sel lnull-lfilter*)
have *2*: *llength nella* = *eSuc* (*llength* (*ltl nella*))
by (*metis 1 llength-LCons*)
have *3*: ¬ *lnull* (*lfilter Pa nella*)
by (*meson a2 a3 lnull-lfilter*)
show *?thesis*
by (*metis 2 3 a4 iless-Suc-eq lfilter-llength-one-conv-a lhd-LCons-ltl llength-LCons llength-LNil*
llist.collapse(*1*) *one-eSuc*)
qed
qed

lemma *nfilter-nlength-zero-conv-c*:
$(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ j \neq k \longrightarrow \neg\ P\ (nnth\ nell\ j))) =$
$(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ j{<}k \vee k{<}j \longrightarrow \neg\ P\ (nnth\ nell\ j)))$
using *antisym-conv3* **by** *auto*

lemma *nfilter-nlength-zero-conv-d*:
$(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ j{<}k \vee k{<}j \longrightarrow \neg\ P\ (nnth\ nell\ j))) \longleftrightarrow$
$(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j.\ j{<}k\ \longrightarrow \neg\ P\ (nnth\ nell\ j)) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ k{<}j \longrightarrow \neg\ P\ (nnth\ nell\ j)))$
by (*meson enat-ord-simps*(*2*) *le-cases le-less-trans*)

lemma *nfilter-nlength-zero-conv-b*:
assumes $(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ j \neq k \longrightarrow \neg\ P\ (nnth\ nell\ j)))$
shows $(\exists x \in nset\ nell.\ P\ x) \wedge nlength(nfilter\ P\ nell) = 0$
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
metis co.enat.exhaust-sel iless-Suc-eq in-lset-conv-lnth llength-eq-0,
metis co.enat.exhaust-sel co.enat.sel(*2*) *iless-Suc-eq lfilter-llength-one-conv-b llength-eq-0 one-eSuc*)

lemma *nfilter-nlength-zero-conv*:
$((\exists x \in nset\ nell.\ P\ x) \wedge nlength(nfilter\ P\ nell) = 0) \longleftrightarrow$
$(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ j \neq k \longrightarrow \neg\ P\ (nnth\ nell\ j)))$
using *nfilter-nlength-zero-conv-a*[*of nell P*] *nfilter-nlength-zero-conv-b*[*of nell P*]
by *blast*

lemma *nfilter-nlength-zero-conv-1*:
$((\exists x \in nset\ nell.\ P\ x) \wedge nlength(nfilter\ P\ nell) = 0) \longleftrightarrow$
$(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ j{<}k \vee k{<}j \longrightarrow \neg\ P\ (nnth\ nell\ j)))$
using *nfilter-nlength-zero-conv*[*of nell P*] *nfilter-nlength-zero-conv-c*[*of nell P*]
by *blast*

lemma *nfilter-nlength-zero-conv-2*:
$((\exists x \in nset\ nell.\ P\ x) \wedge nlength(nfilter\ P\ nell) = 0) \longleftrightarrow$
$(\exists\ k \leq nlength\ nell.\ P\ (nnth\ nell\ k) \wedge$
$\quad (\forall j.\ j{<}k\ \longrightarrow \neg\ P\ (nnth\ nell\ j)) \wedge$
$\quad (\forall j{\leq}\ nlength\ nell.\ k{<}j \longrightarrow \neg\ P\ (nnth\ nell\ j)))$
using *nfilter-nlength-zero-conv-1*[*of nell P*] *nfilter-nlength-zero-conv-d*[*of nell P*]
by *blast*

lemma *nfilter-ndropns-nmap-help-0*:

assumes $\exists\ x \in nset\ nell.\ P\ x$
$j \leq nnth(nkfilter\ P\ 0\ nell)\ 0$
shows $(nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell)) = (nfilter\ P\ (ndropn\ j\ nell))$
using *assms*
proof (*induction j arbitrary*: *nell*)
case *0*
then show *?case*
by (*metis ndropn-0 nfilter-nkfilter-ndropn-1 zero-enat-def zero-le*)
next
case (*Suc j*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
by *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: *is-NNil x22* $\Longrightarrow$
$nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell) = nfilter\ P\ (ndropn\ (Suc\ j)\ nell)$
by (*metis NCons Suc.prems*(*2*) *Suc-le-D ndropn-Suc-NCons ndropn-is-NNil*)
have *2*: $P\ x21 \wedge \neg$*is-NNil x22* $\Longrightarrow$
$nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell) = nfilter\ P\ (ndropn\ (Suc\ j)\ nell)$
using *Suc NCons*
by *simp*
(*metis Suc.prems*(*1*) *dual-order.strict-trans1 nkfilter-not-before nnth-0 zero-less-Suc*)
have *3*: $\neg P\ x21 \wedge \neg$*is-NNil x22* $\Longrightarrow$
$nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell) = nfilter\ P\ (ndropn\ (Suc\ j)\ nell)$
using *Suc NCons* **by** (*simp add*: *nkfilter-nleast*)
show *?thesis*
using *1 2 3* **by** *blast*
qed
qed
qed

lemma *nfilter-nappend-ntaken*:
assumes $\exists\ x \in nset\ (ntaken\ k\ nell).\ P\ x$
$k \leq nlength\ nell$
shows $nfilter\ P\ (ntaken\ k\ nell) =$
$ntaken\ (the\text{-}enat\ (nlength(nfilter\ P\ (ntaken\ k\ nell))))\ (nfilter\ P\ nell)$
using *assms*
apply *transfer*
proof *auto*
show $\bigwedge k\ nell\ P\ x.$
$\neg\ lnull\ nell \Longrightarrow$
$enat\ k \leq epred\ (llength\ nell) \Longrightarrow$
$x \in lset\ (ltake\ (enat\ (Suc\ k))\ nell) \Longrightarrow$
$P\ x \Longrightarrow$
$\forall x{\in}lset\ nell.\ \neg\ P\ x \Longrightarrow$

```
      lfilter P (ltake (enat (Suc k)) nell) =
      ltake (enat (Suc (the-enat (epred (llength (lfilter P (ltake (enat (Suc k)) nell)))))))) nell
  by (metis dual-order.strict-trans1 in-lset-conv-lnth llength-ltake lprefix-lnthD ltake-is-lprefix
    min.cobounded2)
next
 fix k
 fix nell :: 'a llist
 fix P
 fix x
 fix xa
 assume a0: ¬ lnull nell
 assume a1: enat k ≤ epred (llength nell)
 assume a2: x ∈ lset (ltake (enat (Suc k)) nell)
 assume a3: P x
 assume a4: xa ∈ lset nell
 assume a5: P xa
 show lfilter P (ltake (enat (Suc k)) nell) =
      ltake (enat (Suc (the-enat (epred (llength (lfilter P (ltake (enat (Suc k)) nell)))))))) (lfilter P nell)
 proof (cases k = epred (llength nell))
 case True
 then show ?thesis
  proof −
  have f1: eSuc (enat k) = llength nell
  by (simp add: True a0)
  then have llength (lfilter P nell) ≠ 0
  by (metis (no-types) a2 a3 eSuc-enat llength-eq-0 lnull-lfilter ltake-all order-refl)
  then show ?thesis
   using f1 by (metis True co.enat.exhaust-sel eSuc-enat enat-the-enat epred-le-epredI infinity-ileE
   llength-lfilter-ile ltake-all order-refl)
  qed
 next
 case False
 then show ?thesis
  proof −
  have f1: llength nell ≠ 0
  using a0 llength-eq-0 by blast
  have g1: ¬ lnull (lfilter P (ltake (enat (Suc k)) nell))
  by (meson a2 a3 lnull-lfilter)
  then show ?thesis
  using f1
  proof −
   have f1: enat k < epred (llength nell)
   using False a1 order.not-eq-order-implies-strict by blast
   have f2: ∀ e. e = 0 ∨ e = eSuc (epred e)
   by (metis co.enat.exhaust-sel)
   then have g2: enat (Suc k) < llength nell
   using f1 by (metis (no-types) Suc-ile-eq ‹llength nell ≠ 0› iless-Suc-eq)
   then show ?thesis
   using f2
   using lfilter-lappend-ltake[of Suc k]
```

```
  proof −
   have eSuc (enat k) = min (enat (Suc k)) (llength nell)
   by (metis ‹enat (Suc k) < llength nell› eSuc-enat min.strict-order-iff)
   then have eSuc (enat k) = llength (ltake (enat (Suc k)) nell)
   by simp
   then show ?thesis
   by (metis g1 g2 co.enat.exhaust-sel eSuc-enat eSuc-infinity enat-the-enat infinity-ileE
       lfilter-lappend-ltake llength-eq-0 llength-lfilter-ile)
  qed
 qed
 qed
qed
```

```
lemma nappend-nfilter-nfinite:
 assumes ∃ x ∈ nset (nellx). P x
         ∃ x ∈ nset (nelly). P x
         nfinite nellx
 shows    nfilter P (nappend nellx nelly) = nappend (nfilter P nellx) (nfilter P nelly)
using assms
by transfer auto

lemma nfilter-nappend-ndropn:
 assumes ∃ x ∈ nset (ndropn (Suc k) nell). P x
         ∃ x ∈ nset (ntaken k nell). P x
          (Suc k) ≤ nlength nell
 shows nfilter P (ndropn (Suc k) nell) =
       ndropn (the-enat (eSuc(nlength(nfilter P (ntaken k nell))))) (nfilter P nell)
proof −
 have 1: nfinite (nfilter P (ntaken k nell))
   by (metis Suc-ile-eq assms(3) dual-order.strict-implies-order enat-ile length-nfilter-le
       min.absorb1 nfinite-conv-nlength-enat ntaken-nlength)
 have 2: nfilter P nell = nappend (nfilter P (ntaken k nell)) (nfilter P (ndropn (Suc k) nell))
   using assms nappend-nfilter-nfinite[of (ntaken k nell) P (ndropn (Suc k) nell)]
         nappend-ntaken-ndropn[of k nell]  nfinite-ntaken[of k nell] by simp
 have 3: ndropn (the-enat (eSuc(nlength(nfilter P (ntaken k nell)))))
                  (nappend (nfilter P (ntaken k nell)) (nfilter P (ndropn (Suc k) nell)))
          =  (nfilter P (ndropn (Suc k) nell))
   by (metis 1 antisym-conv2 gr-zeroI ile-eSuc iless-Suc-eq less-imp-le ndropn-0
       ndropn-nappend3 nfinite-conv-nlength-enat one-enat-def plus-1-eSuc(2) plus-enat-simps(1)
       the-enat.simps zero-less-diff)
 show ?thesis
   by (simp add: 2 3)
qed

lemma nkfilter-nappend-ntaken:
 assumes ∃ x ∈ nset (ntaken k nell). P x
         k≤ nlength nell
```

shows *nkfilter P n (ntaken k nell) =*
ntaken (the-enat (nlength(nkfilter P n (ntaken k nell)))) (nkfilter P n nell)
using *assms*
apply *transfer*
proof (*auto simp add*: *kfilter-not-lnull-conv kfilter-lnull-conv*)
show $\bigwedge k$ *nell P n x.*
$\neg$ *lnull nell* $\Longrightarrow$
enat k $\leq$ *epred (llength nell)* $\Longrightarrow$
$x \in$ *lset (ltake (enat (Suc k)) nell)* $\Longrightarrow$
P x $\Longrightarrow$
$\forall x{\in}$*lset nell.* $\neg$ *P x* $\Longrightarrow$
kfilter P n (ltake (enat (Suc k)) nell) =
ltake (enat (Suc (the-enat (epred (llength (kfilter P n (ltake (enat (Suc k)) nell)))))))) (iterates Suc n)
by (*meson lset-ltake subsetD*)
next
fix *ka* :: *nat* **and** *nella* :: $'a$ *llist* **and** *Pa* :: $'a \Rightarrow$ *bool* **and** *na* :: *nat* **and** *x* :: $'a$ **and** *xa* :: $'a$
assume *a1*: *Pa x*
assume *a2*: *Pa xa*
assume *a3*: *xa* $\in$ *lset nella*
assume *a4*: *x* $\in$ *lset (ltake (enat (Suc ka)) nella)*
have *f5*: $\bigwedge e$. *e = enat 0* $\vee$ *eSuc (epred e) = e*
by (*metis* (*no-types*) *co.enat.exhaust-sel zero-enat-def*)
have *f6*: $\bigwedge as$. *min* ∞ *(llength (as::*$'a$ *llist)) = llength as*
by *simp*
have *f7*: *eSuc* $\infty = \infty$
by *simp*
have *f8*: *(enat (Suc (the-enat (epred (llength (kfilter Pa na (ltake (enat (Suc ka)) nella)))))))) =*
(llength (kfilter Pa na (ltake (enat (Suc ka)) nella)))
proof $-$
have *f1*: $\forall n$. $\neg$ $\infty \leq$ *enat n*
by (*meson infinity-ileE*)
have $\neg$ *lnull (kfilter Pa na (ltake (enat (Suc ka)) nella))*
by (*metis* (*no-types*) *a1 a4 kfilter-not-lnull-conv*)
then show *?thesis*
using *f1*
by (*metis co.enat.exhaust-sel dual-order.trans eSuc-enat eSuc-ile-mono enat-the-enat kfilter-llength*
llength-eq-0 llength-lfilter-ile llength-ltake min.cobounded1)
qed
moreover
{ assume *llength nella* $\neq \infty$
then have *ltake (enat (Suc ka)) nella = nella* $\longrightarrow$
ltake (llength (lfilter Pa (ltake (enat (Suc ka)) nella))) (kfilter Pa na nella) =
kfilter Pa na (ltake (enat (Suc ka)) nella) $\wedge$
epred (llength (lfilter Pa (ltake (enat (Suc ka)) nella))) ≠ ∞
using *f7 f6 f5 a3 a2* **by** (*metis* (*no-types*) *kfilter-llength llength-eq-0 llength-lfilter-ile*
lnull-lfilter ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq zero-enat-def)
moreover
{ assume *ltake (enat (Suc ka)) nella* $\neq$ *nella*
then have *enat (Suc ka) < llength nella* $\wedge$ *min (enat (Suc ka)) (llength nella)* $\neq \infty \vee$
enat (Suc ka) < llength nella $\wedge$ *llength (lfilter Pa (ltake (enat (Suc ka)) nella))* $\neq$ *eSuc* ∞

```
   by (metis (no-types) enat-ord-code(4) ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq) }
  ultimately have enat (Suc ka) < llength nella ∧ min (enat (Suc ka)) (llength nella) ≠ ∞ ∨
                  enat (Suc ka) < llength nella ∧ llength (lfilter Pa (ltake (enat (Suc ka)) nella)) ≠ eSuc ∞ ∨
                    ltake (llength (lfilter Pa (ltake (enat (Suc ka)) nella))) (kfilter Pa na nella) =
                    kfilter Pa na (ltake (enat (Suc ka)) nella) ∧
                    epred (llength (lfilter Pa (ltake (enat (Suc ka)) nella))) ≠ ∞
  by fastforce }
 ultimately show kfilter Pa na (ltake (enat (Suc ka)) nella) =
                 ltake (enat (Suc (the-enat (epred (llength (kfilter Pa na (ltake (enat (Suc ka)) nella)))))))
                      (kfilter Pa na nella)
 using f6 f5 a4 a1
 kfilter-lappend-ltake[of (enat (Suc ka)) nella Pa na ]
 by (metis enat-ord-code(4) kfilter-llength)
qed

lemma nfilter-ndropns-nmap-help-1:
 assumes ∃ x ∈ nset nell. P x
         j ≤ nnth(nkfilter P 0 nell) (Suc 0)
         nnth (nkfilter P 0 nell) 0 < j
         (Suc 0) ≤ nlength(nfilter P nell)
 shows   (nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell)) =
         (nfilter P (ndropn j nell))
using assms
proof
 (induction j arbitrary: nell )
 case 0
 then show ?case
 by blast
 next
 case (Suc j)
 then show ?case
    proof (cases nell)
    case (NNil x1)
    then show ?thesis
    by auto
    next
    case (NCons x21 x22)
    then show ?thesis
      proof −
       have 1: is-NNil x22 ⟹
                 nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
                 nfilter P (ndropn (Suc j) nell)
         by (metis NCons Suc.prems(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)
       have 2: P x21 ∧ ¬ is-NNil x22 ⟹
                 nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
                 nfilter P (ndropn (Suc j) nell)
         using Suc NCons
         proof simp
         assume a1: P x21 ∧ ¬ is-NNil x22
         assume a2: enat (Suc 0) ≤ nlength (nfilter P (NCons x21 x22))
```

assume *a3*: *nell = NCons x21 x22*
assume *a4*: *Suc j ≤ nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)*
have *f5*: *∀ as p a n. ((∃ a. (a::'a) ∈ nset as ∧ p a) ∨ ¬ p a) ∨*
nkfilter p n (NCons a as) = NNil n
by (*metis* (*no-types*) *nkfilter-NCons-a*)
have *f6*: *∀ as p n. (∀ a. (a::'a) ∉ nset as ∨ ¬ p a) ∨*
nlength (nkfilter p n as) = nlength (nfilter p as)
by (*meson nkfilter-nlength*)
have *f7*: *∀ p as. (∀ a. (a::'a) ∉ nset as ∨ ¬ p a) = (∀ a. a ∉ nset as ∨ ¬ p a)*
by *meson*
have *f8*: *nlength (nkfilter P 0 (NCons x21 x22)) =*
nlength (nfilter P (NCons x21 x22))
by (*meson a1 nellist.set-intros*(*2*) *nkfilter-nlength*)
have *f9*: *∀ p as. (∀ a. (a::'a) ∉ nset as ∨ ¬ p a) = (∀ a. a ∉ nset as ∨ ¬ p a)*
by *meson*
have *f10*: *(∃ a. a ∈ nset x22 ∧ P a) ⟶*
nkfilter P 0 (NCons x21 x22) = NCons 0 (nkfilter P (Suc 0) x22)
using *a1* **by** (*simp add*: *Bex-def-raw*)
then have *f11*: *(∃ a. a ∈ nset x22 ∧ P a) ⟶*
nnth (nkfilter P (Suc 0) x22) 0 − Suc 0 = nnth (nkfilter P 0 x22) 0
using *f9 f8 f7 a2* **by** (*metis Suc-ile-eq iless-Suc-eq nkfilter-nnth-n-zero nlength-NCons*)
have *f14*: *j < nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)*
using *a4 Suc-le-eq* **by** *blast*
have *∃ a. a ∈ nset x22 ∧ P a*
using *f8 f5 a2 a1* **by** (*metis Suc-ile-eq gr-implies-not-zero nlength-NNil*)
then have *(∃ a. a ∈ nset x22 ∧ P a) ∧ nfilter P (ndropn (nnth (nkfilter P 0 x22) 0) x22) =*
nfilter P (ndropn j x22)
using *f14 f11 a4*
proof −
have *j ≤ nnth (nkfilter P 0 x22) 0*
using ‹*∃ a. a ∈ nset x22 ∧ P a*› *f10 f11 f14* **by** *fastforce*
then show *?thesis*
by (*simp add*: *Bex-def-raw* ‹*∃ a. a ∈ nset x22 ∧ P a*› *nfilter-ndropns-nmap-help-0*)
qed
then show *nfilter P (ndropn (nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)) (NCons x21 x22)) =*
nfilter P (ndropn j x22)
using *f11 a4*
by (*metis One-nat-def Suc-le-D diff-Suc-1 f10 ndropn-Suc-NCons nnth-Suc-NCons*)
qed
have *3*: *¬P x21 ∧ ¬ is-NNil x22 ⟹*
nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
nfilter P (ndropn (Suc j) nell)
using *Suc NCons*
proof *simp*
assume *a1*: *⋀nell. ⟦∃ a∈nset nell. P a; j ≤ nnth (nkfilter P 0 nell) (Suc 0);*
nnth (nkfilter P 0 nell) 0 < j; enat (Suc 0) ≤ nlength (nfilter P nell)⟧ ⟹
nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
nfilter P (ndropn j nell)
assume *a2*: *enat (Suc 0) ≤ nlength (nfilter P x22)*
assume *a3*: *Suc j ≤ nnth (nkfilter P (Suc 0) x22) (Suc 0)*

```
        assume a4: ∃x∈nset x22. P x
        assume a5: nnth (nkfilter P (Suc 0) x22) 0 < Suc j
        have f12: nnth (nkfilter P 0 x22) (Suc 0) =
                        ( (nnth (nkfilter P (Suc 0) x22) (Suc 0)) − (Suc 0))
        using nkfilter-nnth-n-zero[of x22 P Suc 0 Suc 0 ]
        by (simp add: a2 a4 nkfilter-nlength)
        then have f13: j ≤ nnth (nkfilter P 0 x22) (Suc 0)
        using a3 by linarith
        have f14: enat 0 ≤ nlength (nfilter P x22)
        using a2 by (metis One-nat-def one-enat-def order.trans zero-enat-def zero-le-one)
        have f15: nlength (nkfilter P (Suc 0) x22) = nlength (nfilter P x22)
        by (simp add: a4 nkfilter-nlength)
        then have Suc 0 ≤ nnth (nkfilter P (Suc 0) x22) 0
        by (simp add: a4 f14 nkfilter-lowerbound)
        then have Suc (nnth (nkfilter P 0 x22) 0) ≤ j
        by (metis a4 a5 add-Suc leD nkfilter-nleast not-less-eq-eq)
        then have nfilter P (ndropn (nnth (nkfilter P 0 x22) (Suc 0)) x22) =
                    nfilter P (ndropn j x22)
        by (simp add: Suc.IH Suc-le-lessD a2 a4 f13)
        then show nfilter P (ndropn (nnth (nkfilter P (Suc 0) x22) (Suc 0)) (NCons x21 x22)) =
                    nfilter P (ndropn j x22)
        using ndropn-Suc-NCons
        by (metis One-nat-def a2 a4 f12 f15 le-add-diff-inverse nkfilter-lowerbound plus-1-eq-Suc)
      qed
    show ?thesis
      using 1 2 3 by blast
  qed
 qed
qed

lemma nfilter-ndropns-nmap-help-j:
 assumes ∃ x ∈ nset nell. P x
         j ≤ nnth(nkfilter P 0 nell) (Suc i)
         nnth (nkfilter P 0 nell) i < j
         (Suc i) ≤ nlength(nfilter P nell)
 shows   (nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell)) =
         (nfilter P (ndropn j nell))
using assms
proof (induction j arbitrary: nell i)
case 0
then show ?case
by blast
next
case (Suc j)
then show ?case
  proof (cases nell)
  case (NNil x1)
  then show ?thesis
  by simp
  next
```

case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: *is-NNil x22* ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
nfilter P (*ndropn* (*Suc j*) *nell*)
by (*metis NCons Suc.prems*(*2*) *Suc-le-D ndropn-Suc-NCons ndropn-is-NNil*)
have *2*: ¬ *is-NNil x22* ∧ *i* = *0* ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
nfilter P (*ndropn* (*Suc j*) *nell*)
using *Suc nfilter-ndropns-nmap-help-1*[*of nell P*] **by** *metis*
have *3*: *P x21* ∧ ¬ *is-NNil x22* ∧ *0*< *i* ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) = *nfilter P* (*ndropn* (*Suc j*) *nell*)
using *Suc NCons*
proof *simp*
assume *a1*: ⋀*nell i*. ⟦∃ *a*∈*nset nell*. *P a*; *j* ≤ *nnth* (*nkfilter P 0 nell*) (*Suc i*);
nnth (*nkfilter P 0 nell*) *i* < *j*; *enat* (*Suc i*) ≤ *nlength* (*nfilter P nell*)⟧ ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
nfilter P (*ndropn j nell*)
assume *a2*: *P x21* ∧ ¬ *is-NNil x22* ∧ *0* < *i*
assume *a3*: *enat* (*Suc i*) ≤ *nlength* (*nfilter P* (*NCons x21 x22*))
assume *a4*: *nell* = *NCons x21 x22*
assume *a5*: *Suc j* ≤ *nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*)
assume *a6*: *nnth* (*nkfilter P 0* (*NCons x21 x22*)) *i* < *Suc j*
show *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*)) (*NCons x21 x22*)) =
nfilter P (*ndropn j x22*)
proof −
have *f0*: ∃ *a*∈*nset x22*. *P a*
by (*metis a2 a3 dual-order.antisym enat.inject nat.distinct*(*1*) *nfilter-NCons-a*
nlength-NNil zero-enat-def zero-le)
have *f1*: *nlength* (*nkfilter P 0* (*NCons x21 x22*)) = *nlength* (*nfilter P* (*NCons x21 x22*))
using *a4* **by** (*metis* (*no-types*) *Suc*(*2*) *nkfilter-nlength*)
have *f2*: *nkfilter P 0* (*NCons x21 x22*) = *NCons 0* (*nkfilter P* (*Suc 0*) *x22*)
by (*metis a2 a3 dual-order.antisym enat.inject f1 nat.distinct*(*1*) *nkfilter-NCons*
nkfilter-NCons-a nlength-NNil zero-enat-def zero-le)
have *f3*: *nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*) =
nnth ((*nkfilter P* (*Suc 0*) *x22*)) (*i*)
by (*simp add*: *f2*)
have *f4*: *enat i* ≤ *nlength* (*nkfilter P* (*Suc 0*) *x22*)
by (*metis Suc-ile-eq a3 f1 f2 iless-Suc-eq nlength-NCons*)
have *f5*: *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*) = (*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*)
using *nkfilter-nnth-n-zero*[*of x22 P i Suc 0*]
using *a5 f0 f3 f4* **by** *linarith*
have *f6*: *ndropn* (*nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*)) (*NCons x21 x22*) =
ndropn ((*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*)) (*NCons x21 x22*)
using *f3 f5* **by** *presburger*
have *f7*: *ndropn* ((*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*)) (*NCons x21 x22*) =
ndropn (*nnth* ((*nkfilter P 0 x22*)) (*i*)) *x22*
using *ndropn-Suc-NCons* **by** *auto*
have *f8*: *j* ≤ *nnth* (*nkfilter P 0 x22*) (*i*)

```
    using a5 f3 f5 by linarith
   have f11: nnth (nkfilter P 0 (NCons x21 x22)) i  = nnth ( (nkfilter P (Suc 0) x22)) (i−1)
    by (metis One-nat-def Suc-pred a2 f2 nnth-Suc-NCons)
   have f12: enat (i − 1) ≤ nlength (nkfilter P (Suc 0) x22)
    by (metis One-nat-def Suc-ile-eq Suc-pred a2 f4 less-imp-le)
   have f13: (Suc 0) ≤ nnth ( (nkfilter P (Suc 0) x22)) (i−1)
    by (meson f0 f12 nkfilter-lowerbound)
   have f14: nnth ( (nkfilter P (Suc 0) x22)) (i−1) = (Suc 0) +nnth ( (nkfilter P 0 x22)) (i−1)
    using nkfilter-nnth-n-zero[of x22 P i−1 Suc 0 ] f12 f0  f13 by (metis le-add-diff-inverse)
   have f9: nnth (nkfilter P 0 x22) (i−1) < j
    using a6 f11 f14 by linarith
   have f10:  i ≤ nlength (nfilter P x22)
    by (metis f0 f4 nkfilter-nlength)
   show ?thesis using a1[of x22 i−1]
    by (metis Suc-diff-1 a2 f0 f10 f3 f5 f7 f8 f9)
 qed
qed
have 4: ¬P x21 ∧ ¬ is-NNil x22 ∧ 0< i ⟹
        nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
        nfilter P (ndropn (Suc j) nell)
  using Suc NCons
  proof simp
   assume a0: ¬ P x21 ∧ ¬ is-NNil x22 ∧ 0 < i
   assume a2: ⋀nell i. ⟦∃ a∈nset nell. P a; j ≤ nnth (nkfilter P 0 nell) (Suc i);
                    nnth (nkfilter P 0 nell) i < j; enat (Suc i) ≤ nlength (nfilter P nell)⟧ ⟹
              nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
              nfilter P (ndropn j nell)
   assume a1: ∃ x∈nset x22. P x
   assume a4: Suc j ≤ nnth (nkfilter P (Suc 0) x22) (Suc i)
   assume a5: nnth (nkfilter P (Suc 0) x22) i < Suc j
   assume a3: enat (Suc i) ≤ nlength (nfilter P x22)
   assume a6: nell = NCons x21 x22
   show nfilter P (ndropn (nnth (nkfilter P (Suc 0) x22) (Suc i)) (NCons x21 x22)) =
       nfilter P (ndropn j x22)
   proof −
    have f1: nlength (nkfilter P 0 (NCons x21 x22)) = nlength (nfilter P (NCons x21 x22))
    by (metis NCons Suc.prems(1) nkfilter-nlength)
    have f2: nkfilter P 0 (NCons x21 x22) =  (nkfilter P (Suc 0) x22)
    using NCons Suc.prems(2) a1 a5 by auto
    have f3: nnth (nkfilter P 0 (NCons x21 x22)) (Suc i) =
          nnth (nkfilter P (Suc 0) x22) (Suc i)
    by (simp add: f2)
    have f4: enat (Suc i) ≤ nlength (nkfilter P (Suc 0) x22)
    using NCons Suc.prems(4) f1 f2 by auto
    have f5: nnth ( (nkfilter P (Suc 0) x22)) (Suc i) = (Suc 0) +nnth ( (nkfilter P 0 x22)) (Suc i)
    by (metis One-nat-def Suc-le-D a1 a4 diff-Suc-1 f4 nkfilter-nnth-n-zero plus-1-eq-Suc)
    have f6: (ndropn (nnth (nkfilter P (Suc 0) x22) (Suc i)) (NCons x21 x22)) =
          ndropn ( (Suc 0) +nnth ( (nkfilter P 0 x22)) (Suc i)) (NCons x21 x22)
    by (simp add: f5)
    have f7: ndropn ( (Suc 0) +nnth ( (nkfilter P 0 x22)) (Suc i)) (NCons x21 x22) =
```

```
            ndropn ( nnth ( (nkfilter P 0 x22)) (Suc i)) (x22)
        by simp
        have f8: j ≤ nnth (nkfilter P 0 x22) (Suc i)
        using a4 f5 by force
        have f11: nnth (nkfilter P 0 (NCons x21 x22)) i  = nnth ( (nkfilter P (Suc 0) x22)) (i)
        by (simp add: f2)
        have f12: enat (i ) ≤ nlength (nkfilter P (Suc 0) x22)
        using Suc-ile-eq f4 by auto
        have f13: (Suc 0) ≤ nnth ( (nkfilter P (Suc 0) x22)) (i)
        by (simp add: a1 f12 nkfilter-lowerbound)
        have f14: nnth ( (nkfilter P (Suc 0) x22)) (i) = (Suc 0) +nnth ( (nkfilter P 0 x22)) (i)
        by (metis a1 f12 f13 le-add-diff-inverse nkfilter-nnth-n-zero)
        have f9: nnth (nkfilter P 0 x22) (i) < j
        using a5 f14 by auto
        show ?thesis
        using Suc.IH a1 a3 f6 f7 f8 f9 by presburger
      qed
     qed
     show ?thesis
     using 1 2 3 4 by fastforce
    qed
  qed
qed

lemma nfilter-ndropns-nmap:
 assumes ∃ x ∈ nset (ndropns nell). P x
         (Suc i) ≤ nlength(nfilter P (ndropns nell))
 shows  ndropn (Suc i) (nmap (λs. nnth s 0) (nfilter P (ndropns nell))) =
         (nmap (λs. nnth s 0)
              (nfilter P (ndropns (ndropn (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) nell))))
proof −
 have 1: ndropn (Suc i) (nmap (λs. nnth s 0) (nfilter P (ndropns nell))) =
          nmap (λs. nnth s 0) (ndropn (Suc i) (nfilter P (ndropns nell)))
   by (simp add: ndropn-nmap)
 have 2: (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) ≤ nlength (ndropns nell)
   using assms nkfilter-nkfilter-ntaken[of (ndropns nell) P  i]
   by (metis Suc-ile-eq antisym-conv2 less-imp-le min.orderE nkfilter-nlength not-le-imp-less
       ntaken-all ntaken-nlength)
 have 3: (nfilter P (ndropns (ndropn (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) nell))) =
          (nfilter P (ndropn (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) (ndropns nell)))
   by (simp add: 2 ndropn-ndropns)
 have 4: (ndropn (Suc i) (nfilter P (ndropns nell))) =
          (nfilter P (ndropn (nnth (nkfilter P 0 (ndropns nell)) (Suc i)) (ndropns nell)))
   by (simp add: assms(1) assms(2) nfilter-nkfilter-ndropn-1 nkfilter-nlength)
 have 5:  (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) ≤ (nnth (nkfilter P 0 (ndropns nell)) (Suc i))
   by (simp add: Suc-leI assms(1) assms(2) nidx-nkfilter-expand nkfilter-nlength)
 have 6:  i < nlength(nfilter P (ndropns nell))
   using Suc-ile-eq assms(2) by blast
 have 7: nnth (nkfilter P 0 (ndropns nell)) i < (Suc (nnth (nkfilter P 0 (ndropns nell)) i))
   by simp
```

have *8*: (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0* (*ndropns nell*)) (*Suc i*)) (*ndropns nell*))) =
(*nfilter P* (*ndropn* (*Suc* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *i*)) (*ndropns nell*)))
using *5 6 7 assms*
nfilter-ndropns-nmap-help-j[*of ndropns nell P* (*Suc* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *i*)) *i*]
by *blast*
show *?thesis*
using *1 3 4 8* **by** *auto*
qed

lemma *ndropns-nfilter-nnth-exist-ndropn*:
assumes ∃ *x* ∈ *nset* (*ndropns nell*). *P x*
j ≤ *nlength* (*nfilter P* (*ndropns nell*))
shows (∃ *i*. *enat i* ≤ *nlength nell* ∧ *nnth* (*nfilter P* (*ndropns nell*)) *j* = *ndropn i nell*)
proof −
have *1*: *nnth* (*nfilter P* (*ndropns nell*)) *j* ∈ *nset* (*ndropns nell*)
using *assms in-nset-conv-nnth nfilter-nnth* **by** *fastforce*
show *?thesis* **using** *1***using** *in-nset-ndropns* **by** *blast*
qed

lemma *nfilter-ndropns-nnth-bound*:
assumes (∃ *ys* ∈ *nset* (*ndropns xs*). *P ys*)
j ≤ *nlength* (*nfilter P* (*ndropns xs*))
shows *nlength* ((*nnth* (*nfilter P* (*ndropns xs*)) *j*)) ≤ *nlength xs*
using *assms ndropns-nfilter-nnth-exist-ndropn*[*of xs P j*]
by (*metis add.commute enat.simps*(*3*) *enat-add-sub-same enat-le-plus-same*(*1*) *less-eqE ndropn-nlength*)

lemma *ndropns-nfilter-ndropn*:
assumes (*Suc k*) ≤ *nlength nell*
∃ *x* ∈ *nset* (*ndropns* (*ndropn* (*Suc k*) *nell*)). *P x*
∃ *x*∈*nset* (*ntaken k* (*ndropns nell*)). *P x*
shows (*nfilter P* (*ndropns* (*ndropn* (*Suc k*) *nell*))) =
(*ndropn* (*the-enat* (*eSuc*(*nlength*(*nfilter P* (*ntaken k* (*ndropns nell*)))))) (*nfilter P* (*ndropns nell*)))
using *assms*
ndropn-ndropns[*of Suc k nell*] *ndropns-nlength*[*of nell*] *nfilter-nappend-ndropn*[*of k ndropns nell P*]
by *simp*

lemma *ndropns-nfilter-ndropn-a*:
assumes *k* ≤ *nlength*(*nfilter P* (*ndropns nell*))
∃ *x* ∈ *nset* (*ndropns nell*). *P x*
shows *ndropn k* (*nfilter P* (*ndropns nell*)) =
nfilter P (*ndropns* (*ndropn* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *k*) *nell*))
using *assms ndropn-ndropns nfilter-nkfilter-ndropn-1*[*of ndropns nell P k*]
nkfilter-upperbound[*of ndropns nell P k 0*]
by (*metis* (*mono-tags*, *lifting*) *add.left-neutral nkfilter-nlength zero-enat-def*)

lemma *nfilter-nlength-imp*:
assumes ∃ *x* ∈ *nset nell*. *P x* ∧ *Q x*
shows *nlength* (*nfilter* (λ*x*. *P x* ∧ *Q x*) *nell*) ≤ *nlength* (*nfilter P nell*)
using *assms* **by** *transfer* (*auto simp add*: *lfilter-nlength-imp epred-le-epredI*)

lemma *nkfilter-chop*:
assumes *nlast nellx = nfirst nelly*
P (nlast nellx)
nfinite nellx
shows *nkfilter P k (nfuse nellx nelly) =*
nfuse (nkfilter P k nellx) (nkfilter P (k+(the-enat (nlength nellx))) nelly)
using *assms*
proof (*auto simp add*: *nfuse-def1*)
show *nlast nellx = nfirst nelly ⟹*
P (nfirst nelly) ⟹
nfinite nellx ⟹
is-NNil nelly ⟹
¬ is-NNil (nkfilter P (k + the-enat (nlength nellx)) nelly) ⟹
nkfilter P k nellx = nappend (nkfilter P k nellx) (ntl (nkfilter P (k + the-enat (nlength nellx)) nelly))
by *transfer auto*
show *nlast nellx = nfirst nelly ⟹*
P (nfirst nelly) ⟹
nfinite nellx ⟹ ¬ is-NNil nelly ⟹
is-NNil (nkfilter P (k + the-enat (nlength nellx)) nelly) ⟹
nkfilter P k (nappend nellx (ntl nelly)) = nkfilter P k nellx
apply *transfer*
proof (*auto simp add*: *kfilter-lappend-lfinite lappend-lnull2*)
fix *nellxa* :: *'a llist* **and** *nellya* :: *'a llist* **and** *Pa* :: *'a ⇒ bool* **and** *ka* :: *nat* **and** *b* :: *nat*
assume *a1*: (*if lnull (kfilter Pa (ka + the-enat (epred (llength nellxa))) nellya*)
then iterates Suc (ka + the-enat (epred (llength nellxa)))
else kfilter Pa (ka + the-enat (epred (llength nellxa))) nellya) = LCons b LNil
assume *a2*: *¬ lnull nellya*
assume *a3*: *Pa (lhd nellya)*
assume *a4*: *¬ lnull (kfilter Pa (ka + the-enat (llength nellxa)) (ltl nellya))*
have *f5*: *lnull (LNil::nat llist)*
using *llength-LNil llength-eq-0* **by** *blast*
have *f6*: *nellya = LCons (lhd nellya) (ltl nellya)*
using *a2* **by** *auto*
then have *Pa (lhd nellya)*
using *a3* **by** *auto*
then have *False*
by (*metis a1 a2 a4 eq-LConsD f6 kfilter-code(2) kfilter-lnull-conv llength-LNil llength-eq-0 llist.set-sel(1)*)
then show *lappend (kfilter Pa ka nellxa) (kfilter Pa (ka + the-enat (llength nellxa)) (ltl nellya)) = iterates Suc ka*
by *meson*
next
fix *nellxa* :: *'b llist* **and** *nellya* :: *'b llist* **and** *Pa* :: *'b ⇒ bool* **and** *ka* :: *nat* **and** *b* :: *nat*
assume *a1*: (*if lnull (kfilter Pa (ka + the-enat (epred (llength nellxa))) nellya*)
then iterates Suc (ka + the-enat (epred (llength nellxa)))
else kfilter Pa (ka + the-enat (epred (llength nellxa))) nellya) = LCons b LNil
assume *a2*: *¬ lnull nellya*
assume *a3*: *Pa (lhd nellya)*
assume *a4*: *¬ lnull (kfilter Pa (ka + the-enat (llength nellxa)) (ltl nellya))*
have *f5*: *lnull (LNil::nat llist)*
using *llength-LNil llength-eq-0* **by** *blast*

have *f6*: *nellya* = *LCons* (*lhd nellya*) (*ltl nellya*)
using *a2* **by** *auto*
then have *Pa* (*lhd nellya*)
using *a3* **by** *auto*
then have *False*
using *f6 f5 a4 a1* **by** (*metis kfilter-LCons kfilter-llength-n-zero llength-eq-0 ltl-simps*(*2*) *not-lnull-conv*)
then show *lappend* (*kfilter Pa ka nellxa*) (*kfilter Pa* (*ka* + *the-enat* (*llength nellxa*)) (*ltl nellya*)) =
kfilter Pa ka nellxa
by *meson*
qed
next
show *nlast nellx* = *nfirst nelly* ⟹
P (*nfirst nelly*) ⟹
nfinite nellx ⟹
¬ *is-NNil nelly* ⟹
¬ *is-NNil* (*nkfilter P* (*k* + *the-enat* (*nlength nellx*)) *nelly*) ⟹
nkfilter P k (*nappend nellx* (*ntl nelly*)) =
nappend (*nkfilter P k nellx*) (*ntl* (*nkfilter P* (*k* + *the-enat* (*nlength nellx*)) *nelly*))
apply *transfer*
proof (*auto simp add*: *lappend-iterates kfilter-lnull-conv split*:*if-split-asm*)
show *f1*:⋀*nellx nelly P k x xa*.
¬ *lnull nellx* ⟹
¬ *lnull nelly* ⟹
llast nellx = *lhd nelly* ⟹
P (*lhd nelly*) ⟹
lfinite nellx ⟹
∀ *b*. *nelly* ≠ *LCons b LNil* ⟹
∀ *b*. *kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly* ≠ *LCons b LNil* ⟹
x ∈ *lset nelly* ⟹ *P x* ⟹ ∀ *x*∈*lset nellx*. ¬ *P x* ⟹ *P xa* ⟹ *xa* ∈ *lset* (*ltl nelly*) ⟹
kfilter P k (*lappend nellx* (*ltl nelly*)) = *iterates Suc k*
by (*metis in-lset-lappend-iff lappend-lbutlast-llast-id lbutlast-lfinite llist.set-intros*(*1*))
next
show *f2*:⋀*nellx nelly P k x xa xb*.
¬ *lnull nellx* ⟹
¬ *lnull nelly* ⟹
llast nellx = *lhd nelly* ⟹
P (*lhd nelly*) ⟹
lfinite nellx ⟹
∀ *b*. *nelly* ≠ *LCons b LNil* ⟹
∀ *b*. *kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly* ≠ *LCons b LNil* ⟹
x ∈ *lset nelly* ⟹
P x ⟹
xa ∈ *lset nellx* ⟹
P xa ⟹
P xb ⟹
xb ∈ *lset nellx* ⟹
kfilter P k (*lappend nellx* (*ltl nelly*)) =
lappend (*kfilter P k nellx*) (*ltl* (*kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly*))
proof −
fix *nellxa* :: *'b llist* **and** *nellya* :: *'b llist* **and** *Pa* :: *'b* ⇒ *bool* **and**

ka :: *nat* **and** *x* :: *'b* **and** *xa* :: *'b* **and** *xb* :: *'b*
assume *a1*: ¬ *lnull nellya*
assume *a2*: *Pa* (*lhd nellya*)
assume *a3*: *llast nellxa* = *lhd nellya*
assume *a4*: ¬ *lnull nellxa*
assume *a5*: *lfinite nellxa*
have *f6*: *nellya* = *LCons* (*lhd nellya*) (*ltl nellya*)
using *a1* **by** *auto*
have *f7*: *LCons* (*lhd nellya*) (*ltl nellya*) ≠ *LNil* ∧ *lhd* (*LCons* (*lhd nellya*) (*ltl nellya*)) = *lhd nellya* ∧
ltl (*LCons* (*lhd nellya*) (*ltl nellya*)) = *ltl nellya*
by *auto*
then have *f8*: *kfilter Pa* (*the-enat* (*epred* (*llength nellxa*)) + *ka*) (*LCons* (*lhd nellya*) (*ltl nellya*)) =
LCons (*the-enat* (*epred* (*llength nellxa*)) + *ka*)
(*kfilter Pa* (*Suc* (*the-enat* (*epred* (*llength nellxa*)) + *ka*)) (*ltl nellya*))
using *f6 a2* **by** (*metis* (*no-types*) *kfilter-LCons*)
have *f9*: ∀ *bs p n bsa*. ¬ *lfinite* (*bs*::*'b llist*) ∨ *kfilter p n* (*lappend bs bsa*) =
lappend (*kfilter p n bs*) (*kfilter p* (*n* + *the-enat* (*llength bs*)) *bsa*)
by (*meson kfilter-lappend-lfinite*)
have *f10*: *lfinite* (*lbutlast nellxa*)
using *a5* **by** (*meson lbutlast-lfinite*)
have *f11*: *lappend* (*kfilter Pa ka* (*lbutlast nellxa*))
(*kfilter Pa* (*ka* + *the-enat* (*llength* (*lbutlast nellxa*))) *LNil*) =
kfilter Pa ka (*lbutlast nellxa*)
by *simp*
have *LCons* (*the-enat* (*epred* (*llength nellxa*)) + *ka*) *LNil* =
kfilter Pa (*the-enat* (*epred* (*llength nellxa*)) + *ka*) (*LCons* (*lhd nellya*) *LNil*)
using *a2* **by** *simp*
then have *f12*: *lappend* (*lappend* (*kfilter Pa ka* (*lbutlast nellxa*))
(*kfilter Pa* (*ka* + *the-enat* (*llength* (*lbutlast nellxa*))) *LNil*))
(*LCons* (*the-enat* (*epred* (*llength nellxa*)) + *ka*) *LNil*) = *kfilter Pa ka nellxa*
using *f11 f10 f9 a5 a4 a3* **by** (*metis add.commute lappend-lbutlast-llast-id-lfinite llength-lbutlast*)
have *ltl* (*kfilter Pa* (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*) =
kfilter Pa (*Suc* (*the-enat* (*epred* (*llength nellxa*)) + *ka*)) (*ltl nellya*)
using *f8 f6* **by** (*simp add*: *add.commute*)
then show *kfilter Pa ka* (*lappend nellxa* (*ltl nellya*)) =
lappend (*kfilter Pa ka nellxa*) (*ltl* (*kfilter Pa* (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*))
using *f12 f11 f10 f9 f8 f7 f6 a5 a4 a3*
by (*metis add.commute lappend-lbutlast-llast-id-lfinite lappend-snocL1-conv-LCons2 llength-lbutlast*)
qed
show ⋀*nellx nelly P k x xa xb*.
¬ *lnull nellx* ⟹
¬ *lnull nelly* ⟹
llast nellx = *lhd nelly* ⟹
P (*lhd nelly*) ⟹
lfinite nellx ⟹
∀ *b*. *nelly* ≠ *LCons b LNil* ⟹
∀ *b*. *kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly* ≠ *LCons b LNil* ⟹
x ∈ *lset nelly* ⟹
P x ⟹
xa ∈ *lset nellx* ⟹

```
      P xa ⟹
      P xb ⟹
      xb ∈ lset (ltl nelly) ⟹
      kfilter P k (lappend nellx (ltl nelly)) =
      lappend (kfilter P k nellx) (ltl (kfilter P (k + the-enat (epred (llength nellx))) nelly))
      by (simp add: f2)
  qed
qed

lemma nleast-conv:
 assumes ∃ x ∈ nset nellx. P x
 shows   nleast P nellx = (LEAST na. na ≤ nlength nellx ∧ P (nnth nellx na))
using assms
by transfer
  (auto simp add: min-def lleast-def,
   metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0)

lemma nfilter-chop:
assumes nlast nellx = nfirst nelly
        P (nlast nellx)
       nfinite nellx
shows nfilter P (nfuse nellx nelly) = nfuse (nfilter P nellx)  (nfilter P nelly)
proof (cases is-NNil nelly)
case True
then show ?thesis by (metis nfilter-NNil nfuse-def1 nfuse-leftneutral)
next
case False
then show ?thesis
 proof (cases is-NNil (nfilter P nelly))
 case True
 then show ?thesis unfolding nfuse-def1 using assms nfilter-nappend2[of ntl nelly P nellx]
  nfilter-expand[of nelly P]
  by (auto simp add: nfirst-def nellist.case-eq-if )
   (metis nellist.discI(2) nellist.set-sel(2)  nset-nlast)
 next
 case False
 then show ?thesis
  proof −
   have 1: ∃ x∈nset nellx. P x
     using assms nset-nlast by blast
   have 2: ∃ x∈nset (ntl nelly). P x
     using False assms
     by (metis nellist.collapse(1) nellist.collapse(2) nellist.disc(1) nfilter-NCons-a
       nfilter-NNil nnth-0 ntaken-0 ntaken-nlast)
   have 3: ¬is-NNil (nfilter P nelly) ⟹
            nfilter P (nappend nellx (ntl nelly)) = nappend (nfilter P nellx) (nfilter P (ntl nelly))
     by (simp add: 1 2 assms(3) nappend-nfilter-nfinite)
   have 4: is-NNil (nfilter P nelly) ⟹ nfilter P nellx = nappend (nfilter P nellx) (nfilter P (ntl nelly))
     by (simp add: False)
   have 5: nfilter P (nfuse nellx nelly) = nfilter P  (nappend nellx (ntl nelly))
```

```
    unfolding nfuse-def1
    by (metis (full-types) False nellist.collapse(1) nfilter-NNil)
  have 6: (ntl (nfilter P nelly)) = (nfilter P (ntl nelly))
    using assms 2 False nfilter-expand[of nelly P]
    by (metis in-nset-ntlD nellist.case-eq-if nellist.sel(5) nfirst-def)
  show ?thesis
    by (metis 3 5 6 False nfuse-def1)
 qed
qed
qed


lemma nfilter-chop1:
assumes n≤ nlength nellx
        P (nlast (ntaken n nellx))
shows nfilter P nellx = nfuse (nfilter P (ntaken n nellx)) (nfilter P (ndropn n nellx))
using assms
by (metis nfuse-ntaken-ndropn ndropn-nfirst nfilter-chop nfinite-ntaken ntaken-nlast)

lemma nfilter-chop1-ntaken:
assumes n≤ nlength nellx
        P (nlast (ntaken n nellx))
shows ntaken (the-enat (nlength(nfilter P (ntaken n nellx)))) (nfilter P nellx) =
      (nfilter P (ntaken n nellx))
using assms nfilter-nappend-ntaken by (metis nfinite-ntaken nset-nlast)

lemma nfilter-nlast:
 assumes n≤ nlength nellx
         P (nlast (ntaken n nellx))
 shows nlast(nfilter P (ntaken n nellx)) = (nlast(ntaken n nellx))
using assms
proof (induction n arbitrary: nellx)
case 0
then show ?case by simp
next
case (Suc n)
then show ?case
  proof (cases nellx)
  case (NNil x1)
  then show ?thesis by auto
  next
  case (NCons x21 x22)
  then show ?thesis using Suc
  by auto
    (metis Suc-ile-eq iless-Suc-eq nfilter-NCons nfinite-ntaken nlast-NCons nlength-NCons
     nset-nlast ntaken-Suc-NCons)
  qed
qed

lemma nfilter-nfirst:
```

assumes *P (nfirst(ndropn n nellx))*
shows *nfirst(nfilter P (ndropn n nellx)) = (nfirst (ndropn n nellx))*
using *assms*
proof (*induction n arbitrary*: *nellx*)
case *0*
then show *?case*
by (*auto simp add*: *ndropn-nfirst*)
(*metis nellist.set-intros(1) nfilter-NNil nfilter-nappend-ntaken nlast-NNil nlength-NNil ntaken-0 the-enat-0 zero-enat-def zero-le*)
next
case (*Suc n*)
then show *?case*
by (*metis ndropn-ndropn plus-1-eq-Suc*)
qed

lemma *nfilter-chop1-ndropn*:
assumes *n≤ nlength nellx*
P (nlast (ntaken n nellx))
shows *ndropn (the-enat (nlength(nfilter P (ntaken n nellx)))) (nfilter P nellx) = (nfilter P (ndropn n nellx))*
proof −
have *1*: *(nfilter P nellx) = nfuse (nfilter P (ntaken n nellx)) (nfilter P (ndropn n nellx))*
by (*simp add*: *assms nfilter-chop1*)
have *2*: *nlast(nfilter P (ntaken n nellx)) = nfirst (nfilter P (ndropn n nellx))*
by (*metis assms(1) assms(2) ndropn-nfirst nfilter-nfirst nfilter-nlast ntaken-nlast*)
have *3*: *ndropn (the-enat (nlength(nfilter P (ntaken n nellx)))) (nfuse (nfilter P (ntaken n nellx)) (nfilter P (ndropn n nellx))) = (nfilter P (ndropn n nellx))*
by (*metis 2 assms(1) enat-the-enat infinity-ileE length-nfilter-le min-absorb1 ndropn-nfuse nfinite-conv-nlength-enat ntaken-nlength*)
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *nkfilter-chop1*:
assumes *(enat n) ≤ nlength nellx*
P (nlast (ntaken n nellx))
shows *nkfilter P k nellx = nfuse (nkfilter P k (ntaken n nellx)) (nkfilter P (k+n) (ndropn n nellx))*
using *assms nfuse-ntaken-ndropn[of n nellx] nkfilter-chop[of (ntaken n nellx) (ndropn n nellx) P k]*
by (*metis min.orderE ndropn-nfirst nfinite-ntaken ntaken-nlast ntaken-nlength the-enat.simps*)

lemma *nkfilter-nlast*:
assumes *n≤ nlength nellx*
P (nlast (ntaken n nellx))
shows *nlast(nkfilter P k (ntaken n nellx)) = k+n*
using *assms*
proof (*induction n arbitrary*: *k nellx*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)

then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis*
using *Suc.prems*(*1*) *enat-0-iff*(*1*) **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis*
using *Suc*
proof *auto*
assume *a1*: *P* (*nlast* (*ntaken n x22*))
assume *a2*: $\bigwedge$*nellx k*. ⟦*enat n* ≤ *nlength nellx*; *P* (*nlast* (*ntaken n nellx*))⟧ ⟹
nlast (*nkfilter P k* (*ntaken n nellx*)) = *k* + *n*
assume *enat* (*Suc n*) ≤ *eSuc* (*nlength x22*)
then have *enat n* < *eSuc* (*nlength x22*)
using *Suc-ile-eq* **by** *blast*
then have *f3*: *enat n* ≤ *nlength x22*
by (*meson iless-Suc-eq*)
have ∃ *a*. *a* ∈ *nset* (*ntaken n x22*) ∧ *P a*
using *a1 nfinite-ntaken nset-nlast* **by** *blast*
then show *nlast* (*nkfilter P k* (*NCons x21* (*ntaken n x22*))) = *Suc* (*k* + *n*)
using *f3 a2 a1* **by** (*simp add*: *Bex-def-raw*)
qed
qed
qed

lemma *nkfilter-nfirst*:
assumes *P* (*nfirst*(*ndropn n nellx*))
shows *nfirst*(*nkfilter P k* (*ndropn n nellx*)) = *k*
using *assms*
proof (*induction n arbitrary*: *k nellx*)
case *0*
then show *?case*
by (*metis enat-defs*(*1*) *nellist.set-intros*(*1*) *nkfilter-NNil nkfilter-nappend-ntaken nlength-NNil*
nnth-NNil ntaken-0 the-enat.simps zero-le)
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis* **using** *Suc*
by (*metis ndropn-ndropn plus-1-eq-Suc*)
next
case (*NCons x21 x22*)
then show *?thesis* **using** *Suc*
by *auto*
qed
qed

lemma *nkfilter-chop1-ndropn*:

assumes $n \leq$ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *ndropn* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*)))) (*nkfilter P k nellx*) =
(*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
proof −
have *1*: (*nkfilter P k nellx*) = *nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*simp add*: *assms nkfilter-chop1*)
have *2*: *nlast*(*nkfilter P k* (*ntaken n nellx*)) = *nfirst* (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*metis assms*(*1*) *assms*(*2*) *ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast*)
have *3*: *ndropn* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*))))
(*nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))) =
(*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*metis 2 assms*(*2*) *enat-the-enat infinity-ileE ndropn-nfuse nfinite-conv-nlength-enat*
nfinite-ntaken nlength-nkfilter-le nset-nlast)
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *nkfilter-chop1-ntaken*:
assumes $n \leq$ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *ntaken* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*)))) (*nkfilter P k nellx*) =
(*nkfilter P k* (*ntaken n nellx*))
proof −
have *1*: (*nkfilter P k nellx*) = *nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*simp add*: *assms nkfilter-chop1*)
have *2*: *nlast*(*nkfilter P k* (*ntaken n nellx*)) = *nfirst* (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*metis assms*(*1*) *assms*(*2*) *ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast*)
have *3*: *ntaken* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*))))
(*nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))) =
(*nkfilter P k* (*ntaken n nellx*))
by (*metis 1 assms*(*1*) *assms*(*2*) *nfinite-ntaken nkfilter-nappend-ntaken nset-nlast*)
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *nfilter-nsubn*:
assumes *enat j* ≤ *nlength nellx*
P (*nnth nellx j*)
enat i ≤ *nlength nellx*
P (*nnth nellx i*)
i ≤ *j*
enat ni = (*nlength*(*nfilter P* (*ntaken i nellx*)))
enat nj = (*nlength*(*nfilter P* (*ntaken j nellx*)))
shows (*nfilter P* (*nsubn nellx i j*)) = (*nsubn* (*nfilter P nellx*) *ni nj*)
proof −
have *1*: (*nfilter P* (*nsubn nellx i j*)) = (*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))
by (*simp add*: *nsubn-def1*)
have *2*: (*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*))) =
ntaken (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*))))))
(*nfilter P* (*ndropn i nellx*))
by (*metis assms*(*2*) *assms*(*5*) *enat-ile le-add-diff-inverse2 linorder-le-cases nfilter-chop1-ntaken*

```
    ntaken-all ntaken-ndropn-nlast)
 have 3: ntaken (the-enat (nlength(nfilter P (ntaken (j−i) (ndropn i nellx)))))
              (nfilter P (ndropn i nellx)) =
        ntaken (the-enat (nlength(nfilter P (ntaken (j−i) (ndropn i nellx)))))
              (ndropn (the-enat (nlength(nfilter P (ntaken i nellx)))) (nfilter P nellx))
  by (simp add: assms(3) assms(4) nfilter-chop1-ndropn ntaken-nlast)
 have 4: ntaken (the-enat (nlength(nfilter P (ntaken (j−i) (ndropn i nellx)))))
              (ndropn (the-enat (nlength(nfilter P (ntaken i nellx)))) (nfilter P nellx)) =
     nsubn (nfilter P nellx)
     (the-enat (nlength(nfilter P (ntaken i nellx))))
     ((the-enat (nlength(nfilter P (ntaken (j−i) (ndropn i nellx)))))+
      (the-enat (nlength(nfilter P (ntaken i nellx)))))
  using ntaken-ndropn by blast
 have 5: ((the-enat (nlength(nfilter P (ntaken (j−i) (ndropn i nellx)))))+
          (the-enat (nlength(nfilter P (ntaken i nellx))))) =
         ((the-enat (nlength(nfilter P (nsubn nellx i j))))+
          (the-enat (nlength(nfilter P (ntaken i nellx)))))
  using 1 by auto
 have 6: ntaken j nellx = nfuse (ntaken i nellx) (nsubn nellx i j)
   using nsubn-nfuse[of 0 i j nellx]
   by (simp add: assms(1) assms(5) nsubn-def1)
 have 7: nlength (nfilter P (nfuse (ntaken i nellx) (nsubn nellx i j))) =
          (nlength (nfilter P (ntaken i nellx)) + nlength (nfilter P (nsubn nellx i j)))
    by (simp add: assms(4) ndropn-nfirst nfilter-chop nfuse-nlength nsubn-def1 ntaken-nfirst ntaken-nlast)
 have 70: nfinite (nfilter P (nfuse (ntaken i nellx) (nsubn nellx i j)))
    by (metis 6 assms(1) assms(2) nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast)
 have 71: nfinite (nfilter P (ntaken i nellx))
   by (metis assms(3) assms(4) nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast)
 have 72: nfinite (nsubn nellx i j)
    by (simp add: nsubn-def1)
 have 8: nj =
          ((the-enat (nlength(nfilter P (nsubn nellx i j))))+ ni)
    by (metis 6 7 add.commute assms(6) assms(7) enat-le-plus-same(2) enat-the-enat infinity-ileE
        plus-enat-simps(1) the-enat.simps)
 show ?thesis
 using 1 2 3 4 8 by (metis assms(6) the-enat.simps)
qed

lemma nfilter-nsubn-zero:
 assumes enat j ≤ nlength nellx
         P (nnth nellx j)
 shows (nfilter P (nsubn nellx 0 j)) =
        (nsubn (nfilter P nellx)
               0
               (the-enat (nlength(nfilter P (ntaken j nellx))))
        )

  using assms
  by (simp add: nfilter-chop1-ntaken nsubn-def1 ntaken-nlast)
```

1.3.11 nrev

lemma *is-NNil-nrev*[*simp*]:
is-NNil (*nrev xs*) = *is-NNil xs*
apply *transfer*
by (*simp add*: *is-LEmpty-llength*)

lemma *nrev-NNil*[*simp*]:
nrev (*NNil a*) = (*NNil a*)
apply *transfer*
by *force*

lemma *nrev-NCons*[*simp*]:
nrev (*NCons a xs*) = (*if nfinite xs then nappend* (*nrev xs*) (*NNil a*) *else* (*NCons a xs*))
apply *transfer*
unfolding *lrev-def*
by *auto*
(*metis lfinite-code*(*2*) *list-of-LCons lrev-LCons lrev-def rev.simps*(*2*))

lemma *ntl-nrev*[*simp*]:
nlength xs > *1* ⟹
ntl (*nrev xs*) =
(*if nfinite xs then*
(*nappend* (*ntl*(*nrev* (*ntl xs*))) (*NNil* (*nhd xs*)))
else ntl xs)
apply *transfer*
by (*auto simp add*: *is-LEmpty-llength epred-llength*)
(*simp add*: *llength-ltl*)

lemma *ntl-nrev-one*[*simp*]:
nlength xs = *1* ⟹
ntl (*nrev xs*) = (*NNil* (*nhd xs*))
apply *transfer*
apply (*auto simp add*: *is-LEmpty-llength*)
apply (*metis epred-1 epred-llength lappend-lnull1 llength-eq-0 llength-lrev*)
apply (*simp add*: *epred-llength*)
by (*metis co.enat.sel*(*2*) *eSuc-infinity llength-eq-infty-conv-lfinite*)

lemma *ntl-nrev-zero*[*simp*]:
assumes *nlength xs* = *0*
shows *ntl* (*nrev xs*) = *xs*
by (*metis assms ndropn-0 ndropn-nlast nellist.sel*(*4*) *nlength-eq-enat-nfiniteD nrev-NNil*
the-enat-0 zero-enat-def)

lemma *nmap-nrev*:
nmap f (*nrev xs*) = (*nrev* (*nmap f xs*))
apply *transfer*
by (*simp add*: *lmap-lrev*)

lemma *nfinite-nrev*:
nfinite(*nrev xs*) = *nfinite xs*
apply *transfer*
by *simp*

lemma *nset-nrev*:
nset(*nrev xs*) = *nset xs*
apply *transfer*
by (*simp add*: *lset-lrev*)

lemma *nlength-nrev*[*simp*]:
nlength (*nrev xs*) = *nlength xs*
apply *transfer*
by *simp*

lemma *lnth-lrev-help*:
¬ *lnull xs* ⟹ *lfinite xs* ⟹ *enat i* ≤ *epred* (*llength xs*) ⟹ *lnth* (*lrev xs*) *i* = *lnth xs* (*the-enat* (*min* (*enat* (*the-enat* (*epred* (*llength xs*)) − *i*)) (*epred* (*llength xs*))))
using *lnth-lrev*[*of xs i*]
unfolding *min-def*
apply *auto*
apply (*metis co.enat.exhaust-sel diff-diff-left enat-the-enat epred-enat iless-Suc-eq*
llength-eq-0 llength-eq-infty-conv-lfinite plus-1-eq-Suc the-enat.simps)
by (*metis co.enat.exhaust-sel diff-le-self eSuc-infinity enat-ord-simps*(*1*) *enat-the-enat*
llength-eq-0 llength-eq-infty-conv-lfinite)

lemma *nnth-nrev*:
assumes *nfinite xs*
(*enat i*) ≤ *nlength xs*
shows (*nnth* (*nrev xs*) *i*) = (*nnth xs* ((*the-enat* (*nlength xs*)) −*i*))
using *assms*
apply *transfer*
apply *auto*
using *lnth-lrev-help* **by** *blast*

lemma *nappend-nrev-nfinite*:
assumes *nfinite xs*
nfinite ys
shows *nrev* (*nappend xs ys*) = *nappend* (*nrev ys*) (*nrev xs*)
using *assms*
apply *transfer*
by (*simp add*: *lappend-lrev-lfinite*)

lemma *nappend-nrev-infa*:
assumes *nfinite xs*
¬*nfinite ys*
shows *nrev* (*nappend xs ys*) = *nappend xs ys*
using *assms*
apply *transfer*

by *simp*

lemma *nappend-nrev--infb*:
assumes ¬*nfinite xs*
shows *nrev* (*nappend xs ys*) = *nappend* (*nrev xs*) (*nrev ys*)
using *assms*
apply *transfer*
by (*simp add*: *lappend-inf*)

lemma *nrev-nrev*:
nrev (*nrev xs*) = *xs*
apply *transfer*
by (*simp add*: *lrev-lrev*)

lemma *nrev-ntaken*:
assumes *nfinite xs*
enat k ≤ *nlength xs*
shows *nrev* (*ntaken k xs*) = *ndropn* (*the-enat*(*nlength xs*) −*k*) (*nrev xs*)
using *assms*
proof (*transfer*)
fix *xs* :: *'a llist*
fix *k*
show ¬ *lnull xs* ∧ *xs* = *xs* ⟹
lfinite xs ⟹
enat k ≤ *epred* (*llength xs*) ⟹
¬ *lnull*
(*if lfinite* (*ltake* (*enat* (*Suc k*)) *xs*)
then lrev (*ltake* (*enat* (*Suc k*)) *xs*)
else ltake (*enat* (*Suc k*)) *xs*) ∧
(*if lfinite* (*ltake* (*enat* (*Suc k*)) *xs*)
then lrev (*ltake* (*enat* (*Suc k*)) *xs*)
else ltake (*enat* (*Suc k*)) *xs*) =
ldropn
(*the-enat*
(*min* (*enat*
(*the-enat* (*epred* (*llength xs*)) − *k*))
(*epred*
(*llength*
(*if lfinite xs then lrev xs*
else xs)))))
(*if lfinite xs then lrev xs else xs*)
proof −
assume *a0*: ¬ *lnull xs* ∧ *xs* = *xs*
assume *a1*: *lfinite xs*
assume *a2*: *enat k* ≤ *epred* (*llength xs*)
show
¬ *lnull*
(*if lfinite* (*ltake* (*enat* (*Suc k*)) *xs*) *then lrev* (*ltake* (*enat* (*Suc k*)) *xs*)
else ltake (*enat* (*Suc k*)) *xs*) ∧
(*if lfinite* (*ltake* (*enat* (*Suc k*)) *xs*) *then lrev* (*ltake* (*enat* (*Suc k*)) *xs*) *else ltake* (*enat* (*Suc k*)) *xs*) =

```
    ldropn
     (the-enat
       (min (enat (the-enat (epred (llength xs)) − k)) (epred (llength (if lfinite xs then lrev xs else xs)))))
     (if lfinite xs then lrev xs else xs)
 proof −
 have 1: enat (Suc k) ≠ 0
   by (metis eSuc-enat zero-ne-eSuc)
 have 2:  llength xs = 1 ⟹ ?thesis
   using a0 a1 a2 by (simp add: 1 ltake-all one-enat-def)
 have 3:  enat (Suc k) < llength xs ⟹
          (the-enat (min (enat (the-enat (epred (llength xs)) − k)) (epred (llength xs)))) =
          (enat (the-enat (llength xs) − Suc k))
    using a0 a1 a2 by simp
  (metis diff-diff-left diff-le-self enat-ord-simps(1) enat-the-enat epred-enat
   llength-eq-infty-conv-lfinite min.absorb1 plus-1-eq-Suc the-enat.simps)

 have 4: enat (Suc k) < llength xs ⟹ ?thesis
   using a0 a1 a2  lrev-ltake[of xs Suc k]   apply simp using 3
   by (metis 1 ldrop-enat llength-lrev lnull-ldrop lnull-lrev ltake.disc(2))
 have 5:  llength xs =  enat (Suc k) ⟹ ?thesis
    using a0 a1 a2 by (simp add: 1 ltake-all)
 have 6:  llength xs = 1 ∨ enat (Suc k) < llength xs ∨ llength xs =  enat (Suc k)
   using a0 a1 a2 apply simp
    by (metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0 order-neq-le-trans)
 show ?thesis using a0 1 2 4 5  6 by blast
 qed
 qed
qed


lemma nrev-ndropn:
 assumes nfinite xs
         enat k ≤ nlength xs
 shows  nrev (ndropn k xs) = ntaken (the-enat(nlength xs) −k) (nrev xs)
using assms
proof transfer
fix xs :: 'a llist
fix k :: nat
show ¬ lnull xs ∧ xs = xs ⟹
       lfinite xs ⟹
       enat k ≤ epred (llength xs) ⟹
       ¬ lnull
           (if lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
            then lrev (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
            else ldropn (the-enat (min (enat k) (epred (llength xs)))) xs) ∧
       (if lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
        then lrev (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
        else ldropn (the-enat (min (enat k) (epred (llength xs)))) xs) =
       ltake (enat (Suc (the-enat (epred (llength xs)) − k))) (if lfinite xs then lrev xs else xs)
proof −
```

assume *a0*: ¬ *lnull xs* ∧ *xs* = *xs*
assume *a1*: *lfinite xs*
assume *a2*: *enat k* ≤ *epred* (*llength xs*)
show ¬ *lnull*
(*if lfinite* (*ldropn* (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
then lrev (*ldropn* (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
else ldropn (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*) ∧
(*if lfinite* (*ldropn* (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
then lrev (*ldropn* (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
else ldropn (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*) =
ltake (*enat* (*Suc* (*the-enat* (*epred* (*llength xs*)) − *k*))) (*if lfinite xs then lrev xs else xs*)
proof −
have *1*: ¬ *lnull*
(*if lfinite* (*ldropn* (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
then lrev (*ldropn* (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
else ldropn (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
using *a0 a1 a2* **apply** *simp*
by (*metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0*)
have *2*: *lfinite* (*ldropn* (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) *xs*)
using *a1 lfinite-ldropn* **by** *blast*
have *3*: (*the-enat* (*min* (*enat k*) (*epred* (*llength xs*)))) = *enat k*
using *a0 a1 a2* **by** *simp*
have *4*: (*enat* (*the-enat* (*llength xs*) − *k*)) = (*enat* (*Suc* (*the-enat* (*epred* (*llength xs*)) − *k*)))
using *a0 a1 a2* **apply** *simp*
using *1 llength-eq-infty-conv-lfinite* **by** *fastforce*
have *5*: *llength xs* = *1* ∨ *enat* (*Suc k*) < *llength xs* ∨ *llength xs* = *enat* (*Suc k*)
using *a0 a1 a2* **apply** *simp*
by (*metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0 order-neq-le-trans*)
have *6*: *llength xs* = *1* ⟹ *?thesis*
using *a0 a1 a2* **by** (*simp add*: *ltake-all one-enat-def*)
have *7*: *enat* (*Suc k*) < *llength xs* ⟹ *?thesis*
using *lrev-ldrop*[*of xs k*] *a0 a1 a2* **apply** *simp*
by (*metis 4 Suc-ile-eq dual-order.strict-iff-order ldrop-enat linorder-not-less*)
have *8*: *llength xs* = *enat* (*Suc k*) ⟹ *?thesis*
using *a0 a1 a2* **apply** *simp*
by (*metis 4 a2 add-diff-cancel-left′ cancel-comm-monoid-add-class.diff-cancel eSuc-enat epred-enat iless-Suc-eq ldrop-enat lrev-ldrop plus-1-eq-Suc the-enat.simps*)
show *?thesis*
using *5 6 7 8* **by** *fastforce*
qed
qed
qed

lemma *nrev-nsubn*:
assumes *nfinite xs*
enat i ≤ *enat j*
enat j ≤ *nlength xs*
shows *nrev* (*nsubn xs i j*) = *nsubn* (*nrev xs*)((*the-enat*(*nlength xs*)) −*j*) ((*the-enat*(*nlength xs*)) − *i*)

using *assms*
proof *transfer*
fix *xs* :: *'a llist*
fix *i j*
assume *a0*: ¬ *lnull xs* ∧ *xs* = *xs*
assume *a1*: *lfinite xs*
assume *a2*: *enat i* ≤ *enat j*
assume *a3*: *enat j* ≤ *epred* (*llength xs*)
show ¬ *lnull*
(*if lfinite* (*lsubc i j xs*)
then lrev (*lsubc i j xs*)
else lsubc i j xs) ∧
(*if lfinite* (*lsubc i j xs*)
then lrev (*lsubc i j xs*) *else lsubc i j xs*) =
lsubc (*the-enat* (*epred* (*llength xs*)) − *j*)
(*the-enat* (*epred* (*llength xs*)) − *i*)
(*if lfinite xs then lrev xs else xs*)
proof −
have *1*: ¬ *lnull* (*if lfinite* (*lsubc i j xs*) *then lrev* (*lsubc i j xs*) *else lsubc i j xs*)
using *a0 a1 a2 a3* **apply** *simp*
by (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lsubc zero-ne-eSuc*)
have *2*: *lfinite* (*lsubc i j xs*)
using *a0 a1 a2 a3* **by** (*simp add*: *lsubc-def*)
have *3*: (*the-enat* (*epred* (*llength xs*)) − *j*) = (*the-enat* (*llength xs*) − (*j* + *1*))
using *a0 a1 a2 a3 lfinite-llength-enat* **by** *fastforce*
have *4*: (*the-enat* (*epred* (*llength xs*)) − *i*) = (*the-enat* (*llength xs*) − (*i* + *1*))
using *a0 a1 a2 a3 lfinite-llength-enat* **by** *fastforce*
have *5*: *enat j* < *llength xs*
using *a0 a1 a2 a3* **apply** *simp*
by (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0*)
have *6*: *lrev* (*lsubc i j xs*) = *lsubc* (*the-enat* (*epred* (*llength xs*)) − *j*)
(*the-enat* (*epred* (*llength xs*)) − *i*)
(*if lfinite xs then lrev xs else xs*)
using *lrev-lsubc*[*of xs i j*] *5 a0 a1 a2 a3 3 4* **by** *presburger*
show *?thesis* **by** (*meson 1 2 6*)
qed
qed

lemma *nlast-nrev*:
shows *nlast* (*nrev xs*) = (*if nfinite xs then nfirst xs else nlast xs*)
apply *transfer*
by (*simp add*: *lfirst-def llast-lrev*)

lemma *nfirst-nrev*:
shows *nfirst* (*nrev xs*) = (*if nfinite xs then nlast xs else nfirst xs*)
apply *transfer*
apply *simp*
by (*metis lfirst-def lfirst-lrev*)

lemma *nellist-all2-nrev*:
nellist-all2 P (*nrev xs*) (*nrev ys*) ⟷ *nellist-all2 P xs ys*
apply *transfer*
using *llist-all2-lfiniteD llist-all2-lrev* **apply** *simp*
by *fastforce*

end

1.3.12 Lsum

lemma *all-nfinite-nnth-a*:
assumes ∀ *i* ≤*nlength nells*. *nfinite* (*nnth nells i*)
shows *all-nfinite nells*
using *assms*
by (*metis in-nset-conv-nnth*)

lemma *all-nfinite-nnth-b*:
assumes *all-nfinite nells*
shows ∀ *i* ≤*nlength nells*. *nfinite* (*nnth nells i*)
using *assms*
by (*meson in-nset-conv-nnth*)

lemma *all-gr-zero-nnth-a*:
assumes ∀ *i* ≤*nlength nells*. *0* < *nlength* (*nnth nells i*)
shows *all-gr-zero nells*
using *assms*
by (*metis in-nset-conv-nnth*)

lemma *all-gr-zero-nnth-b*:
assumes *all-gr-zero nells*
shows ∀ *i* ≤*nlength nells*. *0* < *nlength* (*nnth nells i*)
using *assms*
by (*meson in-nset-conv-nnth*)

lemma *all-nfinite-nnth-ntaken*:
assumes *all-nfinite nells*
n ≤ *nlength nells*
shows *all-nfinite* (*ntaken n nells*)
using *assms*
by (*meson nset-ntaken subsetD*)

lemma *all-nfinite-nnth-ndropn*:
assumes *all-nfinite nells*
n ≤ *nlength nells*

shows *all-nfinite* (*ndropn n nells*)
using *assms*
by (*metis nset-ndropn subsetD*)

lemma *all-nfinite-nnth-nsubn*:
assumes *all-nfinite nells*
$n \leq$ *nlength nells*
$k \leq n$
shows *all-nfinite* (*nsubn nells k n*)
using *assms*
by (*metis in-mono nset-ndropn nset-ntaken nsubn-def1*)

lemma *all-gr-zero-nnth-ntaken*:
assumes *all-gr-zero nells*
$n \leq$ *nlength nells*
shows *all-gr-zero* (*ntaken n nells*)
using *assms*
by (*meson nset-ntaken subsetD*)

lemma *all-gr-zero-nnth-ndropn*:
assumes *all-gr-zero nells*
$n \leq$ *nlength nells*
shows *all-gr-zero* (*ndropn n nells*)
using *assms*
by (*metis nset-ndropn subsetD*)

lemma *all-gr-zero-nnth-nsubn*:
assumes *all-gr-zero nells*
$n \leq$ *nlength nells*
$k \leq n$
shows *all-gr-zero* (*nsubn nells k n*)
using *assms*
by (*metis in-mono nset-ndropn nset-ntaken nsubn-def1*)

lemma *lsum-NNil*:
lsum (*NNil nell*) $a =$ *NNil* (a+(*the-enat* (*nlength nell*)))
by *simp*

lemma *lsum-addzero-NNil*:
assumes *nfinite nell*
shows *addzero* (*lsum* (*NNil nell*) *0*) =
(*if nlength nell* = *0 then* (*NNil 0*) *else* (*NCons 0* (*NNil* (*the-enat* (*nlength nell*)))))
using *assms* **unfolding** *addzero-def*
by *simp*
(*metis less-nat-zero-code ndropn-0 ndropn-nfirst ndropn-nlast nlength-NNil nnth-NNil*)

lemma *lsum-eq-NNil-conv*:
(*lsum nells a*) = (*NNil b*) $\longleftrightarrow$ *is-NNil nells* $\wedge$ a+ (*the-enat* (*nlength*(*nfirst nells*))) = b
by (*metis NNil-eq-ntake-iff lsum.disc-iff*(*1*) *lsum-code*(*1*) *nellist.collapse*(*1*) *nellist.disc*(*1*)

nellist.inject(*1*))

lemma *lsum-eq-NCons-conv*:
lsum nells a = (*NCons b nells1*) ⟷
(∃ *nell nells'*. *nells* = (*NCons nell nells'*) ∧ *b* = *a*+(*the-enat* (*nlength nell*)) ∧
nells1 = *lsum nells'* (*a*+(*the-enat* (*nlength nell*))))
by (*cases nells*) (*simp-all*, *blast*)

lemma *lsum-addzero-NCons*:
addzero (*lsum* (*NCons nell nells*) *0*) = (*NCons 0* (*lsum* (*NCons nell nells*) *0*))
by (*simp add*: *addzero-def*)

lemma *lsum-nfirst*:
nfirst (*lsum nells a*) = *a*+(*the-enat* (*nlength*(*nfirst nells*)))
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **by** *simp* (*metis NNil-eq-ntake-iff nellist.inject*(*1*))
next
case (*NCons x21 nells1*)
then show *?thesis* **by** *simp* (*metis NNil-eq-ntake-iff nnth-0 nnth-NNil ntaken-0 ntaken-nlast*)
qed

lemma *nfinite-lsum-conv-a*:
assumes *nfinite nells*
shows *nfinite* (*lsum nells a*)
using *assms*
proof (*induction nells arbitrary*: *a rule*:*nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons x nells*)
then show *?case* **by** *simp*
qed

lemma *nfinite-lsum-conv-b*:
assumes *nfinite* (*lsum nells a*)
shows *nfinite nells*
using *assms*
proof (*induct zs*≡*lsum nells a arbitrary*: *a nells rule*:*nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis is-NNil-imp-nfinite lsum-eq-NNil-conv*)
next
case (*NCons x nell*)
then show *?case*

by (*metis is-NNil-imp-nfinite lsum-code*(*2*) *nellist.collapse*(*2*) *nellist.sel*(*5*) *nfinite-NConsI*)
qed

lemma *nfinite-lsum-conv*:
nfinite (*lsum nells a*) ⟷ *nfinite nells*
using *nfinite-lsum-conv-a nfinite-lsum-conv-b* **by** *blast*

lemma *lsum-nlength*:
nlength (*lsum nells a*) = *nlength nells*
proof (*cases nfinite nells*)
case *True*
then show *?thesis*
proof (*induction nells arbitrary*: *a rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons x nell*)
then show *?case* **by** *simp*
qed
next
case *False*
then show *?thesis*
proof (*coinduction arbitrary*: *nells a rule*: *enat-coinduct*)
case (*Eq-enat nells1*)
then show *?case*
proof −
have *1*: (*nlength* (*lsum nells1 a*) = (*0*::*enat*)) = (*nlength nells1* = (*0*::*enat*))
by (*metis Eq-enat nfinite-lsum-conv-b nlength-eq-enat-nfiniteD zero-enat-def*)
show *?thesis*
by (*metis 1 Eq-enat nbutlast-not-nfinite nfinite-lsum-conv-b nlength-nbutlast*)
qed
qed
qed

lemma *lsum-addzero-nfirst*:
nfirst (*addzero* (*lsum nells 0*)) = *0*
by (*metis addzero-def ndropn-nfirst ndropn-nfuse nellist.disc*(*1*) *nellist.disc*(*2*) *nfuse-leftneutral*
nfuse-nappend nlast-NNil nlength-NNil nnth-0 the-enat-0)

lemma *lsum-addzero-nlength*:
assumes *nfinite*(*nfirst nells*)
shows (*nlength nells* = *0* ∧ *nlength*(*nfirst nells*) = *0* ⟶
nlength (*addzero* (*lsum nells 0*)) = *0*)
∧
(*nlength nells* = *0* ∧ *nlength*(*nfirst nells*) > *0* ⟶
nlength (*addzero* (*lsum nells 0*)) = *1*)

```
  ∧
  (nlength nells > 0  ⟶
   nlength (addzero (lsum nells 0)) =  (nlength nells) +1)

using assms
by (simp add: addzero-def eSuc-plus-1 lsum-nfirst lsum-nlength)
   (metis enat.distinct(2) enat-the-enat nfinite-nlength-enat zero-enat-def)

lemma sum-nnth-help:
assumes  i>0
         i ≤ nlength nells +1
shows  (∑ k = 0..(i−1). nlength ( nnth (nells) k)) =
       (∑ k = 1..i. nlength ( nnth (nells) (k−1)))
using assms
proof
  (induct i)
  case 0
  then show ?case by blast
  next
  case (Suc i)
  then show ?case
  proof (cases i)
  case 0
  then show ?thesis by simp
  next
  case (Suc nat)
  then show ?thesis
     proof −
      have 1: (0::nat) < (i::nat)
       by (simp add: Suc)
      have 2: enat i ≤ nlength (nells::'a nellist nellist) + (1::enat)
       using Suc.prems(2) Suc-ile-eq by auto
      have 3: (∑ k::nat = 0::nat..i − (1::nat). nlength (nnth nells k)) =
              (∑ k::nat = 1::nat..i. nlength (nnth nells (k − (1::nat))))
       using 1 2 Suc.hyps by blast
      have 4: (∑ k::nat = 0::nat..Suc i − (1::nat). nlength (nnth nells k)) =
              (∑ k::nat = 0::nat..i. nlength (nnth nells k))
        by simp
      have 5: (∑ k::nat = 0::nat..i. nlength (nnth nells k)) =
                nlength (nnth nells i) + (∑ k::nat = 0::nat..(i−1). nlength (nnth nells k))
       by (simp add: Suc)
      have 6:  nlength (nnth nells i) + (∑ k::nat = 1::nat..i. nlength (nnth nells (k − (1::nat)))) =
                (∑ k::nat = 1::nat..Suc i. nlength (nnth nells (k − (1::nat))))
              by simp
      show ?thesis
      using 3 4 5 6 by presburger
    qed
  qed
qed
```

```
lemma lsum-nnth:
 assumes i≤ nlength nells
         all-nfinite nells
 shows nnth (lsum nells a) i = a+(∑ k::nat= 0..i. (the-enat (nlength(nnth nells k))))
using assms
proof
 (induct i arbitrary: a nells)
 case 0
 then show ?case
   proof (cases nells)
   case (NNil x1)
   then show ?thesis by (simp add: nnth-NNil)
   next
   case (NCons x21 x22)
   then show ?thesis by simp
   qed
 next
 case (Suc i)
 then show ?case
    proof (cases nells)
    case (NNil x1)
    then show ?thesis using Suc.prems enat-0-iff(1) by simp
    next
    case (NCons x21 x22)
    then show ?thesis
     proof −
      have 1: nnth (lsum nells a) (Suc i) = nnth (lsum x22 (a + (the-enat (nlength x21)))) i
        by (simp add: NCons)
      have 2:  enat (i::nat) ≤ nlength x22
          using NCons Suc.prems Suc-ile-eq by auto
      have 20: all-nfinite x22
        by (simp add: NCons Suc.prems(2))
      have 3:  nnth (lsum x22 (a + (the-enat (nlength x21)))) i =
              (a + (the-enat (nlength x21))) +
                (∑ k::nat = 0::nat..i. (the-enat (nlength (nnth x22 k))))
        using Suc.hyps[of x22 (a + (the-enat (nlength x21)))] 2  20 by blast
      have 4: a + (∑ k::nat = 0::nat..Suc i. (the-enat (nlength (nnth nells k)))) =
              a + (the-enat ((nlength (nnth nells 0)))) +
                 (∑ k::nat = 1::nat..Suc i. (the-enat (nlength (nnth nells k))))
         by (simp add: sum.atLeast-Suc-atMost)
      have 5: (nlength (nnth nells 0)) = nlength x21
        by (simp add: NCons)
      have 6: (∑ k::nat = 1::nat..Suc i. (the-enat (nlength (nnth nells k)))) =
              (∑ k::nat = 0::nat.. i. (the-enat (nlength (nnth nells (k+1)))))
          using sum.shift-bounds-cl-nat-ivl[of λk. (the-enat (nlength (nnth nells k))) 0 1 i ]
         by simp
      have 7: (∑ k::nat = 0::nat.. i. (the-enat (nlength (nnth nells (k+1))))) =
              (∑ k::nat = 0::nat.. i. (the-enat (nlength (nnth x22 (k))))
        using NCons by auto
```

show *?thesis*
using *1 3 4 5 6 7* **by** *presburger*
qed
qed
qed

lemma *lsum-addzero-nnth*:
assumes $i \leq$ *nlength* (*addzero* (*lsum nells 0*))
nfinite (*nfirst nells*)
shows (*nlength nells* = *0* $\wedge$ *nlength*(*nfirst nells*) = *0* $\longrightarrow$
nnth (*addzero* (*lsum nells 0*)) *i* = (*nnth* (*lsum nells 0*) *i*))
$\wedge$
(*nlength nells* = *0* $\wedge$ *nlength*(*nfirst nells*) > *0* $\longrightarrow$
nnth (*addzero* (*lsum nells 0*)) *i* = (*nnth* (*NCons 0* (*lsum nells 0*))) *i*)
$\wedge$
(*nlength nells* > *0* $\longrightarrow$
nnth (*addzero* (*lsum nells 0*)) *i* = (*nnth* (*NCons 0* (*lsum nells 0*)) *i*))

using *assms* **using** *lsum-addzero-nlength*[*of nells*] *lsum-nlength*[*of nells 0*]
lsum-nfirst[*of nells 0*] **using** *addzero-def* **by** *auto*

lemma *lsum-nlast*:
assumes *nfinite nells*
all-nfinite nells
shows *nlast* (*lsum nells a*) = *a*+($\sum$ *k*::*nat*= *0*..(*the-enat* (*nlength nells*)). (*the-enat* (*nlength*(*nnth nells k*))))
using *assms*
by (*metis* (*no-types*, *lifting*) *enat.simps*(*3*) *enat-le-plus-same*(*2*) *enat-the-enat gen-nlength-def*
lsum-nlength lsum-nnth nfinite-lsum-conv-a nfinite-nlength-enat nlength-code nnth-nlast sum.cong)

lemma *lsum-addzero-nlast*:
assumes *nfinite nells*
shows *nlast* (*addzero* (*lsum nells 0*)) = *nlast*(*lsum nells 0*)
by (*simp add*: *addzero-def*)

lemma *lsum-nnth-nfinite*:
assumes *i* $\leq$ *nlength nells*
all-gr-zero nells
all-nfinite nells
shows ($\sum$ *k*::*nat*= *0*..*i*. (*the-enat* (*nlength*(*nnth nells k*)))) < ∞
using *assms*
using *enat-ord-code*(*4*) **by** *blast*

lemma *sum-finite*:
assumes *all-nfinite nells*
i$\leq$ *nlength nells*
shows ($\sum$ *k*= *0*..*i*. (*nlength*(*nnth nells k*))) < ∞

using *assms*
proof (*induction i arbitrary*: *nells*)
case *0*
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *0* **by** (*simp add*: *nfinite-nlength-enat nnth-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis* **using** *0* **using** *nfinite-nlength-enat* **by** *simp blast*
qed
next
case (*Suc i*)
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *Suc* **by** *simp* (*metis enat.inject old.nat.distinct*(*2*) *zero-enat-def*)
next
case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: $(\sum k = 0..Suc\ i.\ nlength\ (nnth\ nells\ k)) =$
$nlength\ (nnth\ nells\ 0) + (\sum k = 1..Suc\ i.\ nlength\ (nnth\ nells\ k))$
by (*metis One-nat-def sum.atLeast0-atMost-Suc-shift sum.atLeast-Suc-atMost-Suc-shift*)
have *2*: $(\sum k = 1..Suc\ i.\ nlength\ (nnth\ nells\ k)) =$
$(\sum k = 0..\ i.\ nlength\ (nnth\ nells\ (k+1)))$
using *sum.shift-bounds-cl-nat-ivl*[*of* $\lambda k\ .\ (nlength\ (nnth\ nells\ k))\ 0\ 1\ i$]
by (*metis One-nat-def Suc-eq-plus1*)
have *3*: $(\sum k = 0..\ i.\ nlength\ (nnth\ nells\ (k+1))) = (\sum k = 0..\ i.\ nlength\ (nnth\ x22\ k))$
using *NCons* **by** *auto*
have *4*: *all-nfinite x22*
by (*simp add*: *NCons Suc.prems*(*1*))
have *5*: $i \leq$ *nlength x22*
using *NCons Suc.prems*(*2*) *Suc-ile-eq* **by** *auto*
have *6*: $(\sum k = 0..\ i.\ nlength\ (nnth\ x22\ k)) < \infty$
using *Suc.IH*[*of x22*] *4 5* **by** *blast*
have *7*: $nlength\ (nnth\ nells\ 0) < \infty$
by (*metis Suc.prems*(*1*) *all-nfinite-nnth-b enat.distinct*(*2*) *enat-ord-simps*(*4*)
nfinite-nlength-enat zero-enat-def zero-le)
have *8*: $nlength\ (nnth\ nells\ 0) + (\sum k = 0..\ i.\ nlength\ (nnth\ x22\ k)) < \infty$
using *6 7* **by** *force*
show *?thesis* **using** *1 2 3 8* **by** *presburger*
qed
qed
qed

lemma *sum-the-enat*:
assumes *all-nfinite nells*
$i \leq$ *nlength nells*
shows $(\sum k = 0..i.\ \ the\text{-}enat(nlength(nnth\ nells\ k))) = the\text{-}enat(\sum k = 0..i.\ (nlength(nnth\ nells\ k)))$

```
using assms
proof (induction i arbitrary: nells)
case 0
then show ?case
  proof (cases nells)
  case (NNil x1)
  then show ?thesis using 0 by simp
  next
  case (NCons x21 x22)
  then show ?thesis using 0 by simp
  qed
next
case (Suc i)
then show ?case
  proof (cases nells)
  case (NNil x1)
  then show ?thesis using Suc by  simp (metis enat-0-iff(2) old.nat.distinct(2))
  next
  case (NCons x21 x22)
  then show ?thesis
    proof −
     have 1: (∑ k = 0..Suc i. the-enat (nlength (nnth nells k))) =
           the-enat(nlength (nnth nells 0)) + (∑ k = 1..Suc i. the-enat (nlength (nnth nells k)))
       by (simp add: sum.atLeast-Suc-atMost)
     have 2: the-enat(nlength (nnth nells 0)) = the-enat(nlength x21)
      using NCons by simp
     have 3: (∑ k = 1..Suc i. the-enat (nlength (nnth nells k))) =
           (∑ k = 0..i. the-enat (nlength (nnth nells (k+1))))
       using sum.shift-bounds-cl-nat-ivl[of λk .the-enat (nlength (nnth nells k)) 0 1 i]
        by (metis One-nat-def Suc-eq-plus1)
     have 4: (∑ k = 0..i. the-enat (nlength (nnth nells (k+1)))) =
           (∑ k = 0..i. the-enat (nlength (nnth x22 k)))
       using NCons by auto
     have 5: all-nfinite x22
      by (simp add: NCons Suc.prems(1))
     have 6: i≤ nlength x22
      using NCons Suc.prems(2) Suc-ile-eq by auto
     have 7: (∑ k = 0..i. the-enat (nlength (nnth x22 k))) =
           the-enat(∑ k = 0..i. (nlength (nnth x22 k)))
      using Suc.IH[of x22]  5 6 by blast
     have 8: the-enat(∑ k = 0..i. (nlength (nnth x22 k))) =
           the-enat(∑ k = 0..i. (nlength (nnth nells (k+1))))
         using NCons by auto
     have 9: the-enat(∑ k = 0..i. (nlength (nnth nells (k+1)))) =
           the-enat(∑ k = 1..Suc i. (nlength (nnth nells k)))
       using sum.shift-bounds-cl-nat-ivl[of λk . (nlength (nnth nells k)) 0 1 i]
        by (metis One-nat-def Suc-eq-plus1)
     have 10: (∑ k = 1..Suc i. (nlength (nnth nells k))) < ∞
       proof −
        have 100: (∑ k = 1..Suc i. (nlength (nnth nells k))) =
```

```
                (∑ k = 0.. i. (nlength (nnth nells (k+1))))
              using sum.shift-bounds-cl-nat-ivl[of λk . (nlength (nnth nells k)) 0 1 i]
              by (metis One-nat-def Suc-eq-plus1)
          have 101: (∑ k = 0.. i. (nlength (nnth nells (k+1)))) =
                (∑ k = 0.. i. (nlength (nnth x22 k)))
            using NCons by auto
          have 102: (∑ k = 0.. i. (nlength (nnth x22 k))) < ∞
            using sum-finite[of x22 i] 5 6 by blast
          show ?thesis
          using 100 101 102 by presburger
        qed
      have 11: ∃ m . (enat m) = (∑ k = 1..Suc i. (nlength (nnth nells k)))
        by (metis 10 less-infinityE)
       obtain m where 12: (enat m) = (∑ k = 1..Suc i. (nlength (nnth nells k)))
        using 11 by blast
      have 13: ∃ n. (enat n) = (nlength (nnth nells 0))
         by (metis Suc.prems(1) all-nfinite-nnth-b nfinite-nlength-enat zero-enat-def zero-le)
      obtain n where 14: (enat n) = (nlength (nnth nells 0))
        using 13 by blast
      have 15: the-enat(nlength (nnth nells 0)) + the-enat(∑ k = 1..Suc i. (nlength (nnth nells k))) =
            the-enat(nlength (nnth nells 0) + (∑ k = 1..Suc i. (nlength (nnth nells k))))
        by (metis 12 14 plus-enat-simps(1) the-enat.simps)
      have 16: the-enat(nlength (nnth nells 0)) + the-enat(∑ k = 1..Suc i. (nlength (nnth nells k))) =
            the-enat(∑ k = 0..Suc i. (nlength (nnth nells k)))
        using sum.atLeast-Suc-atMost[of 0 Suc i λk. nlength (nnth nells k) ]
        by (metis 15 One-nat-def zero-le)
      show ?thesis
      using 1 16 3 4 7 8 9 by presburger
    qed
  qed
qed



lemma lsum-nnth-leq-Suc:
 assumes i < nlength nells
         all-gr-zero nells
         all-nfinite nells
        a < ∞
 shows  nnth (lsum nells a) i < nnth (lsum nells a) (Suc i)
proof −
 have 1: nnth (lsum nells a) i = a+(∑ k::nat= 0..i. (the-enat (nlength(nnth nells k))))
    using assms less-imp-le-nat lsum-nnth by (metis order-less-imp-le )
 have 2: nnth (lsum nells a) (Suc i) = a+(∑ k::nat= 0..(Suc i). (the-enat (nlength(nnth nells k))))
   using assms Suc-ile-eq lsum-nnth by blast
 have 3: (∑ k::nat= 0..(Suc i). (the-enat (nlength(nnth nells k)))) =
        (∑ k::nat= 0..i. (the-enat (nlength(nnth nells k)))) + (the-enat (nlength(nnth nells (Suc i))))
     using sum.atLeast0-atMost-Suc by blast
 have 4:  nlength(nnth nells (Suc i)) > 0
   using assms by (metis eSuc-enat ileI1 in-nset-conv-nnth)
```

```
 have 5: (∑ k::nat= 0..i. (the-enat (nlength(nnth nells k)))) < ∞
   using lsum-nnth-nfinite[of i nells] assms order.order-iff-strict by blast
 have 6: nlength(nnth nells (Suc i)) < ∞
      using assms
      by (metis all-nfinite-nnth-b  eSuc-enat enat.distinct(2) enat-ord-simps(4) ileI1 nfinite-nlength-enat)
 have 7: a+ (∑ k::nat= 0..i. (the-enat (nlength(nnth nells k)))) <
         a+ (∑ k::nat= 0..(Suc i). (the-enat (nlength(nnth nells k))))
    using 4 5 assms 6 enat-the-enat zero-enat-def by fastforce
 show ?thesis
 using 1 2 7 by presburger
qed


lemma lsum-addzero-nnth-leq-Suc:
 assumes i < nlength(addzero (lsum nells 0))
         all-gr-zero nells
         all-nfinite nells
 shows  nnth (addzero (lsum nells 0)) i < nnth (addzero (lsum nells 0)) (Suc i)
using assms
proof (cases i)
case 0
then show ?thesis
   proof (cases nells)
   case (NNil x1)
   then show ?thesis using 0 assms unfolding addzero-def  by simp
   (metis enat-0-iff(2) ndropn-nfirst ndropn-nlast nfinite-code(1) nfinite-nlength-enat nlast-NNil
    nlength-NNil nnth-nlast the-enat.simps)
   next
   case (NCons x21 x22)
   then show ?thesis using 0 assms unfolding addzero-def by simp
   (metis eSuc-infinity enat-the-enat gr-zeroI ndropn-eq-NNil ndropn-nlast not-eSuc-ilei0
    zero-enat-def)
   qed
   next
case (Suc nat)
then show ?thesis
  proof (cases nells)
  case (NNil x1)
  then show ?thesis
  using Suc addzero-def assms(1) enat-0-iff(1) by fastforce
  next
  case (NCons x21 x22)
  then show ?thesis
    proof −
     have 1: nat = 0 ⟹ ?thesis
       using Suc assms unfolding addzero-def by auto
          (simp add: Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc,
          metis Extended-Nat.eSuc-mono NCons One-nat-def assms(1) assms(2) enat-ord-code(4)
          lsum-addzero-NCons lsum-nlength lsum-nnth-leq-Suc nlength-NCons one-eSuc one-enat-def
          zero-enat-def)
```

have *2*: *nat>0* $\Longrightarrow$ *?thesis*
using *Suc assms* **unfolding** *addzero-def* **by** *auto*
(*simp add*: *Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc*,
metis NCons Suc-ile-eq assms(1) assms(2) enat-ord-code(4) iless-Suc-eq lsum-addzero-NCons
lsum-nlength lsum-nnth-leq-Suc nlength-NCons)
show *?thesis* **using** *1 2* **by** *blast*
qed
qed
qed

lemma *lsum-nidx*:
assumes *all-gr-zero nells*
all-nfinite nells
shows *nidx* (*lsum nells a*)
using *assms* **unfolding** *nidx-expand*
by (*simp add*: *Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc*)

lemma *lsum-addzero-nidx*:
assumes *all-gr-zero nells*
all-nfinite nells
shows *nidx* (*addzero* (*lsum nells 0*))
using *assms* **unfolding** *nidx-expand*
using *Suc-ile-eq lsum-addzero-nnth-leq-Suc* **by** *blast*

lemma *nfusecat-nlength-b*:
assumes *nlastnfirst nells*
all-nfinite nells
1$\leq$ *i*
i$\leq$ *nlength nells*
j $\leq$ *nlength*(*nnth nells i*)
nfinite nells
all-gr-zero nells
shows (*nnth* ((*lsum nells 0*)) (*i*$-$*1*) + *j* $\leq$ *nlength* (*nfusecat nells*))
proof $-$
have *0*: *i*$-$*1* $\leq$ *nlength nells*
by (*metis Suc-ile-eq assms*(*3*) *assms*(*4*) *le-add-diff-inverse order-less-imp-le plus-1-eq-Suc*)
have *1*: (*nnth* ((*lsum nells 0*)) (*i*$-$*1*) = ($\sum$ *k*::*nat*= *0*..(*i*$-$*1*). *the-enat*(*nlength*(*nnth nells k*))))
using *0 lsum-nnth*[*of i*$-$*1 nells 0*] *assms* **by** *presburger*
have *2*: *nlength* (*nfusecat nells*) = ($\sum$ *k*::*nat*= *0*..(*the-enat*(*nlength nells*)). *nlength*(*nnth nells k*))
using *assms nfusecat-nlength-nfinite* **by** *metis*
have *20*: ($\sum$ *k*::*nat*= *0*..(*i*). *the-enat*(*nlength*(*nnth nells k*))) =
($\sum$ *k* = *0*..*i* $-$ *1*. *the-enat* (*nlength* (*nnth nells k*))) +
the-enat (*nlength* (*nnth nells* (*Suc* (*i* $-$ *1*))))
using *assms sum.atLeast0-atMost-Suc*[*of* λ*k*. *the-enat*(*nlength*(*nnth nells k*)) *i*$-$*1*] **by** *simp*
have *21*: ($\sum$ *k* = *0*..*i* $-$ *1*. *the-enat* (*nlength* (*nnth nells k*))) $<$ ∞
using *enat-ord-code*(*4*) **by** *blast*
have *22*: *the-enat* (*nlength* (*nnth nells* (*Suc* (*i* $-$ *1*)))) $<$ ∞
by *auto*

have *23*: $(\sum k{::}nat{=}\ 0..(i).\ the\text{-}enat(nlength(nnth\ nells\ k))) < \infty$
using *enat-ord-simps(4)* **by** *blast*
have *3*: $(\sum k{::}nat{=}\ 0..(i{-}1).\ the\text{-}enat(nlength(nnth\ nells\ k))) + j \leq$
$(\sum k{::}nat{=}\ 0..(i).\ the\text{-}enat(nlength(nnth\ nells\ k)))$
using *20 assms* **by** *simp*
(*metis* (*no-types*, *lifting*) *all-nfinite-nnth-b enat-ord-simps(1) enat-the-enat infinity-ileE*
ndropn-eq-NNil ndropn-nlast)
have *4*: $(\sum k{::}nat{=}\ 0..(i).\ the\text{-}enat(nlength(nnth\ nells\ k))) \leq$
$(\sum k{::}nat{=}\ 0..(the\text{-}enat(nlength\ nells)).\ the\text{-}enat(nlength(nnth\ nells\ k)))$
using *sum.ub-add-nat*[*of 0 i* λ *k*. *the-enat*(*nlength*(*nnth nells k*)) (*the-enat*(*nlength nells*))$-i$]
using *assms(4) assms(6) nfinite-nlength-enat* **by** *fastforce*
have *5*: $(\sum k{::}nat{=}\ 0..(the\text{-}enat(nlength\ nells)).\ the\text{-}enat(nlength(nnth\ nells\ k))) \leq$
$(\sum k{::}nat{=}\ 0..(the\text{-}enat(nlength\ nells)).\ nlength(nnth\ nells\ k))$
using *sum-the-enat*[*of nells* (*the-enat*(*nlength nells*))] *assms*
by (*metis leI less-enatE order-less-imp-not-eq the-enat.simps*)
show *?thesis*
using *1 2 3 4 5* **by** *simp*
(*meson dual-order.trans enat-ord-simps(1)*)
qed

lemma *nfusecat-nlength-b-alt*:
assumes *nlastnfirst nells*
all-nfinite nells
$1 \leq i$
$i \leq$ *nlength nells*
$j \leq$ *nlength*(*nnth nells i*)
$\neg$*nfinite nells*
all-gr-zero nells
shows *nnth* ((*lsum nells 0*)) $(i{-}1) + j \leq$ *nlength* (*nfusecat nells*)
proof $-$
have *1*: $\neg$*nfinite* (*nfusecat nells*)
using *assms* **by** (*metis nfusecat-nfinite*)
show *?thesis*
by (*meson 1 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
qed

lemma *lsum-nfusecat-nnth-lsum-nnth*:
assumes *nlastnfirst nells*
all-gr-zero nells
all-nfinite nells
$i \leq$*nlength nells*
$j \leq$ *nlength*(*nnth nells i*)
shows (*nnth* (*nfusecat nells*) ((*nnth* (*addzero* (*lsum nells 0*)) *i*) $+j$)) = (*nnth* (*nnth nells i*) *j*)
using *assms*
proof (*induction i arbitrary*: *nells j*)
case *0*
then show *?case*
proof (*cases nells*)
case (*NNil x1*)

```
  then show ?thesis using 0 by simp
   (metis add-cancel-right-left addzero-def ndropn-nfirst ndropn-nlast nfinite-NNil nlast-NNil
    nnth-0 nnth-NNil)
  next
  case (NCons x21 x22)
  then show ?thesis using 0 by simp
   (metis 0.prems(1) 0.prems(2) 0.prems(3) add-cancel-right-left addzero-def eSuc-ne-0
      nfirst-nfusecat-nfirst nfuse-nnth nfusecat-NCons nfusecat-nlength-a nlength-NCons nnth-0)
  qed
next
case (Suc i)
then show ?case
  proof (cases nells)
  case (NNil x1)
  then show ?thesis using Suc by (simp add: zero-enat-def)
  next
  case (NCons x21 x22)
  then show ?thesis
   proof −
    have 0: (0::enat) < nlength nells
      by (simp add: NCons)
    have 1: nlastnfirst x22
     using NCons Suc.prems(1) by auto
    have 2: all-gr-zero x22
      using NCons Suc.prems(2) by auto
    have 3: all-nfinite x22
      using NCons Suc.prems(3) by auto
    have 4:   enat (i::nat) ≤ nlength x22
      using NCons Suc.prems(4) Suc-ile-eq by auto
    have 5: nnth (nfusecat nells) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j) =
          nnth (nfuse x21 (nfusecat x22)) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j)
      by (simp add: NCons)
    have 6:  nlength (addzero (lsum nells (0::nat))) = nlength nells + (1::enat)
      using lsum-addzero-nlength[of nells ]
      by (metis NCons lsum-addzero-NCons lsum-nlength nlength-NCons plus-1-eSuc(2))
    have 7: enat (Suc (i::nat) ) ≤ nlength (addzero (lsum (nells::'a nellist nellist) (0::nat)))
      by (simp add: 6 Suc.prems(4) order-less-imp-le plus-1-eSuc(2))
    have 8: (nnth (addzero (lsum nells (0::nat))) (Suc i)) =
          nnth (NCons (0::nat) (lsum nells (0::nat))) (Suc i)
      using lsum-addzero-nnth[of (Suc i) nells ] by (metis NCons lsum-addzero-NCons)
    have 9: nnth (NCons (0::nat) (lsum nells (0::nat))) (Suc i) = nnth (lsum nells 0) i
      by simp
    have 10: nnth (lsum nells 0) i  = (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k)))
      using lsum-nnth[of i nells 0]
      by (metis Suc.prems(3) Suc.prems(4) Suc-ile-eq add-0 order-less-imp-le)
    have 11: nlength x21 ≤ (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) + j
      proof (cases i)
      case 0
      then show ?thesis using NCons by simp
      (metis Suc.prems(3) enat-ord-simps(1) enat-the-enat infinity-ileE le-add1 ndropn-eq-NNil
```

```
 ndropn-nlast nellist.set-intros(2))
 next
 case (Suc nat)
 then show ?thesis
    proof −
     have 12: (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =
               the-enat (nlength (nnth nells 0)) +
               (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k)))
       by (simp add: sum.atLeast-Suc-atMost)
     have 13: (nlength (nnth nells 0)) = nlength x21
       by (simp add: NCons)
     have 14: nlength x21 ≤
               the-enat (nlength x21) +
               (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k))) + j
       using NCons Suc.prems(3) nfinite-nlength-enat by fastforce
     show ?thesis
     using 12 13 14 by presburger
    qed
qed
have 15: nlast x21 = nfirst (nfusecat x22)
 by (metis NCons Suc.prems(1) nfirst-nfusecat-nfirst nlastnfirst-LCons)
have 16: enat (nnth (addzero (lsum nells 0)) (Suc i) + j) ≤ nlength (nfuse x21 (nfusecat x22))
   by (metis 8 9 NCons Suc.prems(1) Suc.prems(2) Suc.prems(3) Suc.prems(4) Suc.prems(5)
       add-diff-cancel-left′ le-add1 nfusecat-NCons nfusecat-nlength-b nfusecat-nlength-b-alt
       plus-1-eq-Suc)
have 17: nlength x21 ≤ enat (nnth (addzero (lsum nells (0::nat))) (Suc i) + j)
    using 10 11 8 9 by presburger
have 18: nnth (nfuse x21 (nfusecat x22)) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j) =
       nnth (nfusecat x22) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j − (the-enat(nlength x21)))
 using nfuse-nnth-var[of (nnth (addzero (lsum nells (0::nat))) (Suc i) + j) x21 nfusecat x22]
 using 15 16 17 by blast
have 19: i=0 ⟹ nnth (addzero (lsum nells (0::nat))) (Suc i) = (the-enat(nlength x21))
    using 8 NCons by auto
have 20: i=0 ⟹ nnth (nfusecat x22) 0 = nnth (nnth x22 i) 0
 using Suc.IH[of x22 0]
 by (metis 1 2 3 4 add.right-neutral all-gr-zero-nnth-b lsum-addzero-NCons
     lsum-addzero-nfirst nnth-0 ntaken-0 ntaken-nlast order-less-imp-le zero-enat-def)
have 21: i=0 ⟹ ?thesis
  by (metis 1 18 19 2 3 4 5 NCons Suc.IH Suc.prems(5) add.commute add.right-neutral
      add-diff-cancel-left′ lsum-addzero-NCons lsum-addzero-nfirst nnth-0 nnth-Suc-NCons ntaken-0
     ntaken-nlast)
have 22: 0<i ⟹ (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =
        (the-enat(nlength x21)) + (∑ k::nat = 0::nat..(i−1). the-enat (nlength (nnth x22 k)))
  proof −
   assume a1: 0<i
   have 23: (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =
           the-enat(nlength x21) + (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k)))
     by (simp add: NCons sum.atLeast-Suc-atMost)
   have 24: (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k))) =
           (∑ k::nat = 0::nat..(i−1). the-enat (nlength (nnth nells (k+1))))
```

```
        using sum.shift-bounds-cl-nat-ivl[of λk .the-enat (nlength (nnth nells k)) 0 1 i−1]
         by (simp add: a1)
       have 25: (∑ k::nat = 0::nat..(i−1). the-enat (nlength (nnth nells (k+1)))) =
             (∑ k::nat = 0::nat..(i−1). the-enat (nlength (nnth x22 (k))))
        using NCons by auto
       show ?thesis
       using 23 24 25 by presburger
     qed
   have 26: 0<i ⟹ nnth (addzero (lsum nells (0::nat))) (Suc i) − (the-enat(nlength x21)) =
                 (∑ k::nat = 0::nat..(i−1). the-enat (nlength (nnth x22 (k))))
    by (simp add: 10 22 8)
   have 260: 0 < nfirst (lsum x22 0)
     proof (cases x22)
     case (NNil x1)
     then show ?thesis by simp
      (metis 2 3 NNil-eq-ntake-iff enat-the-enat gr0I infinity-ileE less-numeral-extra(3)
       ndropn-eq-NNil ndropn-nlast nellist.set-intros(1) nlast-NNil zero-enat-def)
     next
     case (NCons x21 x22)
     then show ?thesis by simp
      (metis 2 3  gr0I  less-numeral-extra(3) ndropn-nfirst ndropn-nfuse ndropn-nlast
       nellist.set-intros(2) nfinite-NNil nfuse-leftneutral nlast-NNil nlength-NNil nnth-0 the-enat-0)
     qed
   have 27: 0<i ⟹ (∑ k::nat = 0::nat..(i−1). the-enat (nlength (nnth x22 (k)))) =
                nnth (addzero (lsum x22 0)) (i)
      unfolding addzero-def using NCons  26 3 4 8 260 apply simp
       using lsum-nnth
       by (metis Suc-ile-eq Suc-pred add-0 nnth-Suc-NCons order-less-imp-le)
   have 28: 0<i ⟹
        nnth (nfusecat x22) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j − (the-enat(nlength x21)))
=
         nnth (nnth x22 i) j
    using Suc.IH[of x22 j]
    by (metis 1 10 2 22 26 27 3 4 8 NCons Nat.add-diff-assoc2 Suc.prems(5)
        le-add1 nnth-Suc-NCons)
   have 29: 0<i ⟹ ?thesis
    using 18 28 NCons by fastforce
   show ?thesis
   using 21 29 by blast
  qed
 qed
qed

lemma lsum-shift:
 assumes nlastnfirst nells
       all-gr-zero nells
       all-nfinite nells
        i≤nlength nells
 shows    nnth (lsum nells a) i = a+ nnth (lsum nells 0) i
using assms by (simp add: lsum-nnth)
```

1.3.13 nsubn_nellist

definition *nsubn-nellist*
where *nsubn-nellist σ l =*
(if nlength l = 0 then (NNil (ndropn (nfirst l) σ))
else nmap (λ (x,y). (nsubn σ x y)) (nzip l (ntl l)))

lemma *nnth-nzip-npair*:
assumes *Suc i ≤ nlength l*
shows *nnth (nzip l (ntl l)) i = (nnth l i, (nnth l (Suc i)))*
using *assms*
by (*metis Suc-ile-eq iless-Suc-eq nellist.exhaust-sel nellist.sel(4) nlength-NCons nnth-ntl nnth-nzip order-less-imp-le*)

lemma *nlength-nzip-npair*:
shows *nlength (nzip l (ntl l)) = nlength (ntl l)*
by (*metis ile-eSuc is-NNil-ntl min.absorb2 min-def nellist.collapse(2) nellist.expand nlast-ntl nlength-NCons nlength-nzip*)

lemma *nnth-nsubn-nellist-NNil*:
assumes *nlength l = 0*
shows *nnth (nsubn-nellist σ l) i = (ndropn (nfirst l) σ)*
using *assms* **unfolding** *nsubn-nellist-def*
by (*simp add*: *nnth-NNil*)

lemma *nnth-nsubn-nellist*:
assumes *(Suc i) ≤ nlength l*
shows *nnth (nsubn-nellist σ l) i = (nsubn σ (nnth l i) (nnth l (Suc i)))*
using *assms*
unfolding *nsubn-nellist-def*
using *nnth-nmap*[*of i nzip l (ntl l) (λ (x,y). (nsubn σ x y))*]
nnth-nzip-npair[*of i l*]
apply *simp*
by (*metis Suc-ile-eq case-prod-conv enat-ord-simps(2) i0-lb ndropn-nmap nnth-nmap nnth-zero-ndropn not-less-eq zero-enat-def*
zero-less-Suc)

lemma *nlength-nsubn-nellist*:
assumes *nlength l > 0*
shows *nlength (nsubn-nellist σ l) = nlength(ntl l)*
using *assms* **unfolding** *nsubn-nellist-def*
using *nlength-nzip-npair* **by** *auto*

lemma *nfinite-nnth-nsubn-nellist-NNil*:
assumes *nidx l*
nlength l = 0
i ≤ nlength l
nfinite σ
shows *nfinite (nnth (nsubn-nellist σ l) i)*
using *assms*

by (*simp add*: *nnth-NNil nsubn-nellist-def*)

lemma *nfinite-nnth-nsubn-nellist*:
assumes *nidx l*
nlength l > *0*
(*Suc i*) ≤ *nlength l*
shows *nfinite* (*nnth* (*nsubn-nellist σ l*) *i*)
using *assms*
by (*simp add*: *nnth-nsubn-nellist nsubn-nfinite*)

lemma *nlength-nnth-nsubn-nellist-NNil*:
assumes *nlength l* = *0*
i ≤ *nlength l*
nfinite σ
shows *nlength* (*nnth* (*nsubn-nellist σ l*) *i*) = *nlength* (*ndropn* (*nfirst l*) *σ*)
using *assms* **by** (*simp add*: *assms*(*2*) *nnth-NNil nsubn-nellist-def*)

lemma *nlength-nnth-nsubn-nellist*:
assumes *nidx l*
(*Suc i*) ≤ *nlength l*
shows *nlength* (*nnth* (*nsubn-nellist σ l*) *i*) =
min (*enat* (*nnth l* (*Suc i*)) − *enat* (*nnth l i*)) (*nlength σ* − *enat* (*nnth l i*))
using *assms*
unfolding *nidx-expand*
by (*simp add*: *nnth-nsubn-nellist nsubn-nlength*)

lemma *nsubn-nellist-ntaken-NNil*:
assumes
nlength l1 = *0*
enat n ≤ *nlength l1*
shows *nsubn-nellist σ* (*ntaken n l1*) = (*NNil* (*ndropn* (*nfirst l1*) *σ*))
using *assms* **unfolding** *nsubn-nellist-def*
by (*simp add*: *ntaken-nfirst*)

lemma *nsubn-nellist-ntaken*:
assumes *nidx l1*
n >*0*
(*Suc n*) ≤ *nlength l1*
nlast l1 ≤ *nlength σ*
nlength l1 > *0*
shows *nsubn-nellist σ* (*ntaken n l1*) =
ntaken (*n*−*1*) (*nsubn-nellist σ l1*)
proof −
have *1* : *nlength* (*nsubn-nellist σ* (*ntaken n l1*)) =
nlength (*ntl* (*ntaken n l1*))
using *assms*
by (*simp add*: *nlength-nsubn-nellist zero-enat-def*)
have *2*: *nlength* (*ntl* (*ntaken n l1*)) = (*n*−*1*)
by (*metis Suc-ile-eq assms*(*2,3*) *bot-nat-0.not-eq-extremum co.enat.sel*(*2*) *eSuc-enat enat-1-iff*(*2*) *epred-enat*

i0-less min.absorb3
nat.inject nellist-split-2-first nlength-NCons ntaken-nlength one-eSuc zero-enat-def)
have *3*: *nlength* (*ntaken* (*n−1*) (*nsubn-nellist* σ *l1*)) = (*n−1*)
by (*metis 2 assms*(*2,5*) *bot-nat-0.not-eq-extremum nlength-nsubn-nellist ntaken-nlength ntl-ntaken*)

have *4*: ⋀ *i*. *i* ≤ *nlength* (*nsubn-nellist* σ (*ntaken n l1*))
⟹
nnth (*nsubn-nellist* σ (*ntaken n l1*)) *i* =
nsubn σ (*nnth l1 i*) (*nnth l1* (*Suc i*))
using *1 2 assms*
by (*simp add*: *Suc-ile-eq nnth-nsubn-nellist ntaken-nnth zero-enat-def*)
have *5*: ⋀ *i*. *i* ≤ *nlength* (*nsubn-nellist* σ (*ntaken n l1*))
⟹
nnth (*ntaken* (*n−1*) (*nsubn-nellist* σ *l1*)) *i* =
nsubn σ (*nnth l1 i*) (*nnth l1* (*Suc i*))
using *assms 1 2 nnth-nsubn-nellist*[*of*] *ntaken-nnth*[*of n−1* (*nsubn-nellist* σ *l1*)]
by (*metis One-nat-def Suc-ile-eq Suc-pred eSuc-enat enat-ord-simps*(*1*) *iless-Suc-eq min.absorb3 min-absorb1*
min-less-iff-conj)
show *?thesis*
by (*metis 1 2 3 4 5 nellist-eq-nnth-eq*)
qed

lemma *nsubn-nellist-ndropn-NNil*:
assumes *nlength l1* = *0*
n ≤ *nlength l1*
nlast l1 ≤ *nlength* σ
shows *nsubn-nellist* σ (*ndropn n l1*) = (*NNil* (*ndropn* (*nfirst l1*) σ))
using *assms*
unfolding *nsubn-nellist-def*
by (*simp add*: *enat-0-iff*(*1*))

lemma *nsubn-nellist-ndropn*:
assumes *nidx l1*
n >*0*
(*Suc n*) ≤ *nlength l1*
nlast l1 ≤ *nlength* σ
nlength l1 > *0*
shows *nsubn-nellist* σ (*ndropn n l1*) =
ndropn (*n*) (*nsubn-nellist* σ *l1*)
proof −
have *01*: *0* < *nlength* (*ndropn n l1*)
using *assms* **apply** *simp*
by (*metis Suc-n-not-le-n add.right-neutral enat-add-sub-same enat-diff-cancel-left enat-ord-simps*(*1*)
le-SucI le-refl
not-infinity-eq order-trans)
have *02*: ∃ *x*. (*ndropn n l1*) = *ndropn* (*Suc n*) (*NCons x l1*)
by *simp*
have *1*: *nlength* (*nsubn-nellist* σ (*ndropn n l1*)) =
nlength (((*ndropn* (*Suc n*) *l1*)))
using *assms nlength-nsubn-nellist*[*of* (*ndropn n l1*) σ] *01 ndropn-Suc-NCons*[*of n - l1*]

by (*metis nellist.sel*(*5*) *ntl-ndropn*)
have *2*: *nlength* (*ndropn* (*n*) (*nsubn-nellist* σ *l1*)) =
nlength (((*ndropn* (*Suc n*) *l1*)))
using *assms ndropn-nlength*[*of n* (*nsubn-nellist* σ *l1*)]
nlength-nsubn-nellist[*of l1* σ]
by (*metis ndropn-Suc-NCons ndropn-nlength nellist-split-2-first*)
have *3*: ⋀*i*. *i*≤ *nlength* (*nsubn-nellist* σ (*ndropn n l1*)) ⟹
nnth (*nsubn-nellist* σ (*ndropn n l1*)) *i* =
nsubn σ (*nnth l1* (*n*+*i*)) (*nnth l1* (*n*+*Suc* (*i*)))
using *ndropn-nnth*[*of n l1*]
using *assms 1 2* **apply** *simp*
by (*metis 1 Suc-ile-eq add-Suc-right iless-Suc-eq ndropn-Suc-conv-ndropn ndropn-nnth nlength-NCons nnth-nsubn-nellist*)
have *4*: ⋀*i*. *i*≤ *nlength* (*nsubn-nellist* σ (*ndropn n l1*)) ⟹
nnth (*ndropn* (*n*) (*nsubn-nellist* σ *l1*)) *i*=
nsubn σ (*nnth l1* (*n*+*i*)) (*nnth l1* (*n*+*Suc* (*i*)))
using *assms 1 2 ndropn-nnth*[*of n* (*nsubn-nellist* σ *l1*)]
apply *simp*
by (*metis 01 Suc-ile-eq add-Suc-right iless-Suc-eq linorder-not-less ndropn-nnth-a nellist-split-2-first nidx-expand nidx-ndropn nle-le*
nlength-NCons nlength-nsubn-nellist nnth-nsubn-nellist)
show *?thesis*
by (*metis 1 2 3 4 nellist-eq-nnth-eq*)
qed

lemma *nsubn-nellist-nappend*:
assumes *nidx l1*
nidx l2
nlength l1 > *0*
nlength l2 > *0*
nfinite l1

nlast l2 ≤ *nlength* σ
shows (*nsubn-nellist* σ (*nappend l1 l2*)) =
(*nappend* (*nsubn-nellist* σ *l1*) (*NCons* (*nsubn* σ (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist* σ *l2*)))
proof −
have *1*: *nlength* (*nsubn-nellist* σ (*nappend l1 l2*)) =
nlength (*ntl* (*nappend l1 l2*))
using *assms nlength-nsubn-nellist*[*of* (*nappend l1 l2*) σ]
by *simp*
have *2*: *nlength* (*ntl* (*nappend l1 l2*)) = *nlength l1* + *nlength l2*
using *assms*
by (*metis enat-add2-eq is-NNil-nappend nellist.collapse*(*2*) *nlength-NCons nlength-nappend one-enat-def plus-1-eSuc*(*2*))
have *3*: *nlength* (*nappend* (*nsubn-nellist* σ *l1*) (*NCons* (*nsubn* σ (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist* σ *l2*)))
=
nlength (*nsubn-nellist* σ *l1*) + *nlength* (*nsubn-nellist* σ *l2*) +*2*
apply *simp*
by (*metis add.commute add.left-commute one-add-one plus-1-eSuc*(*2*))
have *4*: *nlength* (*nsubn-nellist* σ *l1*) + *nlength* (*nsubn-nellist* σ *l2*) +*2* =

nlength l1 + nlength l2
using *assms nlength-nsubn-nellist*[*of l1 σ*] *nlength-nsubn-nellist*[*of l2 σ*]
by (*metis 2 3 One-nat-def ileI1 is-NNil-nappend nappend-ntaken-ndropn nellist-split-2-first nlength-NCons nlength-nappend*
ntl-nappend one-eSuc one-enat-def)
have *5*: ⋀ *i*. *enat i* ≤ *nlength* (*nsubn-nellist σ* (*nappend l1 l2*)) ⟹
nnth (*nsubn-nellist σ* (*nappend l1 l2*)) *i* =
nsubn σ (*nnth* (*nappend l1 l2*) *i*) (*nnth* (*nappend l1 l2*) (*Suc i*))
by (*metis 1 2 Suc-ile-eq iless-Suc-eq nlength-nappend nnth-nsubn-nellist plus-1-eSuc*(*2*))
have *6*: ⋀ *i*. *enat i* < *nlength l1* ⟹
nsubn σ (*nnth* (*nappend l1 l2*) *i*) (*nnth* (*nappend l1 l2*) (*Suc i*)) =
nsubn σ (*nnth l1 i*) (*nnth l1* (*Suc i*))
using *assms*
by (*metis Suc-ile-eq nnth-nappend order-le-less*)
have *7*: ⋀ *i*. *nlength l1* = *enat i* ⟹
nsubn σ (*nnth* (*nappend l1 l2*) *i*) (*nnth* (*nappend l1 l2*) (*Suc i*)) =
nsubn σ (*nnth l1 i*) (*nnth l2 0*)
using *assms*
by (*simp add*: *nnth-nappend*)
have *8*: ⋀ *i*. *nlength l1* < *enat i* ∧ *enat i* ≤ *nlength l1* + *nlength l2* ⟹
nsubn σ (*nnth* (*nappend l1 l2*) *i*) (*nnth* (*nappend l1 l2*) (*Suc i*)) =
nsubn σ (*nnth l2* (*i* − (*Suc* (*the-enat* (*nlength l1*)))))
(*nnth l2* ((*Suc i*) − (*Suc* (*the-enat* (*nlength l1*)))))
using *assms nnth-nappend*[*of l1 l2*] **apply** *auto*
using *Suc-ile-eq* **by** *force*
have *9*: ⋀ *i*. *enat i*< *nlength l1* ⟹
nnth (*nappend* (*nsubn-nellist σ l1*) (*NCons* (*nsubn σ* (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist σ l2*))) *i* =
nsubn σ (*nnth l1 i*) (*nnth l1* (*Suc i*))
using *assms*
by (*metis eSuc-enat ileI1 iless-Suc-eq nellist-split-2-first nlength-NCons nlength-nsubn-nellist nnth-nappend1 nnth-nsubn-nellist*)
have *10*: ⋀ *i*. *enat i* = *nlength l1* ⟹
nnth (*nappend* (*nsubn-nellist σ l1*) (*NCons* (*nsubn σ* (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist σ l2*))) *i*
=
nsubn σ (*nnth l1 i*) (*nnth l2 0*)
using *assms 6 7 9 nnth-nappend*[*of* (*nsubn-nellist σ l1*) (*NCons* (*nsubn σ* (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist σ l2*))]
apply *simp*
using *assms*(*3*)
by (*metis diff-self-eq-0 enat-ord-simps*(*3*) *iless-Suc-eq nellist-split-2-first nfirst-eq-nnth-zero nlength-NCons nlength-nsubn-nellist nnth-0 nnth-nlast the-enat.simps the-enat-eSuc*)
have *11*: ⋀ *i*. *nlength l1* < *enat i* ∧ *enat i* ≤ *nlength l1* + *nlength l2* ⟹
nnth (*nappend* (*nsubn-nellist σ l1*) (*NCons* (*nsubn σ* (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist σ l2*))) *i*
=
nsubn σ (*nnth l2* (*i* − (*Suc* (*the-enat* (*nlength l1*)))))
(*nnth l2* ((*Suc i*) − (*Suc* (*the-enat* (*nlength l1*)))))
proof −
fix *i*
assume *a0*: *nlength l1* < *enat i* ∧ *enat i* ≤ *nlength l1* + *nlength l2*
show *nnth* (*nappend* (*nsubn-nellist σ l1*) (*NCons* (*nsubn σ* (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist σ l2*)))

```
i =
          nsubn σ (nnth l2 (i − (Suc (the-enat (nlength l1)))))
                (nnth l2 ((Suc i) − (Suc (the-enat (nlength l1)))))
      proof −
        have 111: enat i > nlength (nsubn-nellist σ l1) using a0 assms
        by (metis  iless-Suc-eq linorder-not-less nellist-split-2-first nlength-NCons nlength-nsubn-nellist
          order.asym)
        have 112: nnth (nappend (nsubn-nellist σ l1) (NCons (nsubn σ (nlast l1) (nfirst l2)) (nsubn-nellist
σ l2))) i=
                  nnth (NCons (nsubn σ (nlast l1) (nfirst l2)) (nsubn-nellist σ l2))
          (i − Suc (the-enat (nlength (nsubn-nellist σ l1))))
        using assms a0  6 7 8 9 111 nnth-nappend[of (nsubn-nellist σ l1) (NCons (nsubn σ (nlast l1) (nfirst
l2)) (nsubn-nellist σ l2)) i]
         by (meson  linorder-not-less)
        have 113: nnth (NCons (nsubn σ (nlast l1) (nfirst l2)) (nsubn-nellist σ l2))
          (i − Suc (the-enat (nlength (nsubn-nellist σ l1)))) =
                nnth (nsubn-nellist σ l2) (i − (Suc (the-enat (nlength l1))))
          using assms a0 111   nlength-nsubn-nellist
              by (metis Suc-diff-Suc enat-ord-code(4) enat-ord-simps(2) enat-the-enat nellist-split-2-first
nlength-NCons
            nnth-Suc-NCons order.asym the-enat-eSuc)
        have 114: Suc (the-enat (nlength l1)) ≤ i
          using a0   by (metis Suc-le-eq enat-ord-simps(2,3) enat-the-enat  linorder-not-less)
        have 115: enat (Suc (i − Suc (the-enat (nlength l1)))) ≤ nlength l2
          using 114 111 a0 assms apply simp
        by (metis Suc-diff-Suc Suc-le-eq enat-add-sub-same enat-minus-mono1 enat-the-enat idiff-enat-enat
          infinity-ileE le-iff-add   nfinite-conv-nlength-enat)

        have 116: nnth (nsubn-nellist σ l2) (i − (Suc (the-enat (nlength l1)))) =
                  nsubn σ (nnth l2 (i − Suc (the-enat (nlength l1)))) (nnth l2 (Suc (i − Suc (the-enat
(nlength l1)))))

          using   nnth-nsubn-nellist[of  (i − (Suc (the-enat (nlength l1))))  ]
          using 115 by blast
        show ?thesis
        using 112 113 114 116 Suc-diff-le by presburger
       qed
      qed
 have 12: ⋀i .  enat i < nlength l1  ⟹
            nnth (nsubn-nellist σ (nappend l1 l2)) i =
           nnth (nappend (nsubn-nellist σ l1) (NCons (nsubn σ (nlast l1) (nfirst l2)) (nsubn-nellist σ l2)))
i
   using 6 9
   by (metis 1 2 5 enat-le-plus-same(1) linorder-not-less order.asym order-le-less-trans)
 have 13: ⋀i . enat i=nlength l1 ⟹
          nnth (nsubn-nellist σ (nappend l1 l2)) i =
            nnth (nappend (nsubn-nellist σ l1) (NCons (nsubn σ (nlast l1) (nfirst l2)) (nsubn-nellist σ l2)))
i
     using 7 10
     using 1 2 5 enat-le-plus-same(1) by presburger
```

have *14*: $\bigwedge i.\ nlength\ l1 < enat\ i \wedge enat\ i \leq nlength\ l1 + nlength\ l2 \Longrightarrow$
nnth (*nsubn-nellist* σ (*nappend l1 l2*)) *i* =
nnth (*nappend* (*nsubn-nellist* σ *l1*) (*NCons* (*nsubn* σ (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist* σ *l2*)))
i
using *8 11 1 2 5* **by** *auto*
have *15*: *nlength*(*nsubn-nellist* σ (*nappend l1 l2*)) =*nlength l1* + *nlength l2*
using *1 2* **by** *fastforce*
have *16*: $\bigwedge i.\ enat\ i < nlength\ l1 \vee enat\ i$=*nlength l1* $\vee$ (*nlength l1* < *enat i* $\wedge$ *enat i* $\leq$ *nlength l1* + *nlength l2*) $\Longrightarrow$
nnth (*nsubn-nellist* σ (*nappend l1 l2*)) *i* =
nnth (*nappend* (*nsubn-nellist* σ *l1*) (*NCons* (*nsubn* σ (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist* σ *l2*)))
i
using *12 13 14* **by** *fastforce*
have *17*: $\bigwedge\ i.\ i \leq nlength$(*nsubn-nellist* σ (*nappend l1 l2*)) $\Longrightarrow$
nnth (*nsubn-nellist* σ (*nappend l1 l2*)) *i* =
nnth (*nappend* (*nsubn-nellist* σ *l1*) (*NCons* (*nsubn* σ (*nlast l1*) (*nfirst l2*)) (*nsubn-nellist* σ *l2*)))
i
using *1 16 2 linorder-cases* **by** *auto*
show *?thesis*
by (*metis 1 17 2 3 4 nellist-eq-nnth-eq*)
qed

lemma *nfusecat-nappend-nsubn-nellist*:
assumes *nidx l1*
nidx l2
nlength l1 > *0*
nlength l2 > *0*
nfinite l1
nlast l1 = *nfirst l2*
nlast l2 $\leq$ *nlength* σ
shows *nfusecat* (*nappend* (*nsubn-nellist* σ *l1*) (*nsubn-nellist* σ *l2*)) =
nfuse (*nfusecat* (*nsubn-nellist* σ *l1*)) (*nfusecat* (*nsubn-nellist* σ *l2*))
using *assms*
by (*metis nappend-NNil nellist-split-2-first*
nfinite-conv-nlength-enat nfinite-nappend
nfusecat-nappend nlength-nsubn-nellist)

lemma *nlastnfirst-nsubn-nellist-infinite*:
assumes *nidx l*
$\neg$*nfinite l*
$\neg$*nfinite* σ

shows *nlastnfirst* (*nsubn-nellist* σ *l*)
using *assms nlastnfirst-def1*[*of* (*nsubn-nellist* σ *l*)]
nnth-nsubn-nellist[*of* - *l* σ] *nidx-expand*[*of l*] **apply** *auto*
by (*metis Suc-ile-eq add.commute iless-Suc-eq le-add-diff-inverse less-le-not-le nellist-split-2-first nlength-NCons*
nlength-eq-enat-nfiniteD nlength-nsubn-nellist not-gr-zero nsubn-nfirst nsubn-nlast zero-enat-def)

lemma *nlastnfirst-nsubn-nellist-finite*:

assumes *nidx l*
nfinite l
nfinite σ
nlast l ≤ nlength σ
nlength l > 0
shows *nlastnfirst* (*nsubn-nellist σ l*)
using *assms nlastnfirst-def1*[*of* (*nsubn-nellist σ l*)]
nnth-nsubn-nellist[*of* - *l σ*] *nidx-expand*[*of l*] **apply** *auto*
by (*metis Suc-ile-eq add.commute assms*(*5*) *iless-Suc-eq le-add-diff-inverse nellist-split-2-first nlength-NCons nlength-nsubn-nellist*
nsubn-nfirst nsubn-nlast order.strict-iff-order)

lemma *nsubn-nellist-ncons-nfinite*:
assumes *nidx l*

nfirst l > x
nfinite σ
nfinite l
nlast l = (*the-enat*(*nlength σ*))
shows (*nsubn-nellist σ* (*NCons x l*)) =
(*if nlength l* = *0 then* (*NNil* (*nsubn σ x* (*the-enat*(*nlength σ*))))
else (*NCons* (*nsubn σ x* (*nnth l 0*)) (*nsubn-nellist σ l*)))
proof −
have *1*: (*nsubn-nellist σ* (*NCons x l*)) =
(*nmap* (λ(*x*, *y*). *nsubn σ x y*) (*nzip* (*NCons x l*) (*ntl* (*NCons x l*))))
using *assms* **unfolding** *nsubn-nellist-def* **by** *force*
have *2*: (*nmap* (λ(*x*, *y*). *nsubn σ x y*) (*nzip* (*NCons x l*) (*ntl* (*NCons x l*)))) =
(*if nlength l* = *0 then* (*NNil* (*nsubn σ x* (*the-enat*(*nlength σ*))))
else (*NCons* (*nsubn σ x* (*nnth l 0*)) (*nsubn-nellist σ l*)))
proof −
have *20*: (*nzip* (*NCons x l*) (*ntl* (*NCons x l*))) =
(*if nlength l* = *0 then* (*NNil* (*x*, *nfirst l*)) *else* (*NCons* (*x*,*nfirst l*) (*nzip l* (*ntl l*))))
apply *simp*
using *assms*(*4*)
by (*metis i0-less ndropn-0 ndropn-nlast nellist-split-2-first nfirst-eq-nnth-zero nnth-0 nnth-nlast*
nzip-simps(*2*,*3*) *the-enat-0*)
have *21*: (*nmap* (λ(*x*, *y*). *nsubn σ x y*) (*nzip* (*NCons x l*) (*ntl* (*NCons x l*)))) =
(*nmap* (λ(*x*, *y*). *nsubn σ x y*) (*if nlength l* = *0 then* (*NNil* (*x*, *nfirst l*)) *else* (*NCons* (*x*,*nfirst l*) (*nzip l* (*ntl l*)))))
using *20* **by** *auto*
have *22*: (*nmap* (λ(*x*, *y*). *nsubn σ x y*) (*if nlength l* = *0 then* (*NNil* (*x*, *nfirst l*)) *else* (*NCons* (*x*,*nfirst l*) (*nzip l* (*ntl l*))))) =
(*if nlength l* = *0 then* (*NNil* (*nsubn σ x* (*the-enat*(*nlength σ*))))
else (*NCons* (*nsubn σ x* (*nnth l 0*)) (*nsubn-nellist σ l*)))
unfolding *nsubn-nellist-def* **using** *assms* **by** (*auto simp add*: *nfirst-eq-nnth-zero nnth-nlast*)
show *?thesis*
using *21 22* **by** *auto*
qed
show *?thesis*
using *1 2* **by** *auto*

qed

lemma *nsubn-nellist-ncons*:
assumes *nidx l*

nfirst l > x
¬nfinite σ
¬nfinite l
shows (*nsubn-nellist σ* (*NCons x l*)) = (*NCons* (*nsubn σ x* (*nnth l 0*)) (*nsubn-nellist σ l*))
proof −
have *1*: (*nsubn-nellist σ* (*NCons x l*)) = *nmap* (λ(*x*, *y*). *nsubn σ x y*) (*nzip* (*NCons x l*) (*ntl* (*NCons x l*)))

using *assms* **unfolding** *nsubn-nellist-def* **by** *auto*
have *2*: *nmap* (λ(*x*, *y*). *nsubn σ x y*) (*nzip* (*NCons x l*) (*ntl* (*NCons x l*))) =
nmap (λ(*x*, *y*). *nsubn σ x y*) (*NCons* (*x*, *nfirst l*) (*nzip l* (*ntl l*)))
using *assms*
by (*metis is-NNil-imp-nfinite nellist.collapse*(*2*) *nellist.sel*(*5*) *nfirst-eq-nnth-zero nnth-0 nzip-simps*(*3*))
have *3*: *nmap* (λ(*x*, *y*). *nsubn σ x y*) (*NCons* (*x*, *nfirst l*) (*nzip l* (*ntl l*))) =
(*NCons* (*nsubn σ x* (*nnth l 0*)) (*nsubn-nellist σ l*))
unfolding *nsubn-nellist-def*
using *assms* **apply** *simp*
by (*metis nfirst-eq-nnth-zero nlength-eq-enat-nfiniteD zero-enat-def*)
show *?thesis*
using *1 2 3* **by** *auto*
qed

lemma *nfusecat-nsubn-nellist-ncons*:
assumes *nidx l*
nfirst l > x
¬nfinite σ
¬nfinite l
shows *nfusecat* (*nsubn-nellist σ* (*NCons x l*)) =
nfuse (*nsubn σ x* (*nnth l 0*)) (*nfusecat* (*nsubn-nellist σ l*))
using *assms nidx-expand*[*of l*] *nsubn-nellist-ncons*[*of l x σ*]
by *fastforce*

lemma *nfusecat-nsubn-nellist-finite*:
assumes *nfinite l*
nfinite σ
nidx l
nlast l = (*the-enat*(*nlength σ*))
shows *nfusecat* (*nsubn-nellist σ l*) = *ndropn* (*nfirst l*) *σ*
using *assms*
proof (*induction l arbitrary*: *σ rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*simp add*: *nsubn-nellist-def*)
next
case (*NCons x l*)

then show *?case*
proof −
have *1*: (*nsubn-nellist* σ (*NCons* x l)) =
(*if nlength* l = *0 then* (*NNil* (*nsubn* σ x (*the-enat*(*nlength* σ))))
else (*NCons* (*nsubn* σ x (*nnth* l *0*)) (*nsubn-nellist* σ l)))
using *nsubn-nellist-ncons-nfinite*
by (*metis NCons.hyps NCons.prems*(*1,2,3*) *nidx-LCons-1 nlast-NCons*)
have *2*: *nfusecat* (*nsubn-nellist* σ (*NCons* x l)) =
nfusecat(*if nlength* l = *0 then* (*NNil* (*nsubn* σ x (*the-enat*(*nlength* σ))))
else (*NCons* (*nsubn* σ x (*nnth* l *0*)) (*nsubn-nellist* σ l)))
using *1* **by** *auto*
have *3*: *nfusecat*(*if nlength* l = *0 then* (*NNil* (*nsubn* σ x (*the-enat*(*nlength* σ))))
else (*NCons* (*nsubn* σ x (*nnth* l *0*)) (*nsubn-nellist* σ l))) =
(*if nlength* l = *0 then* (*nsubn* σ x (*the-enat*(*nlength* σ)))
else nfuse (*nsubn* σ x (*nnth* l *0*)) (*nfusecat* (*nsubn-nellist* σ l)))
by *simp*
have *4*: *nlength* l = *0* ⟹
(*nsubn* σ x (*the-enat*(*nlength* σ))) = *ndropn* (*nfirst* (*NCons* x l)) σ
by (*metis NCons.prems*(*1*) *idiff-enat-enat ndropn-nlength nfinite-conv-nlength-enat nfirst-eq-nnth-zero nle-le nnth-0 nsubn-def1 ntaken-all*
the-enat.simps)
have *5*: *nlength* l > *0* ⟹
nfuse (*nsubn* σ x (*nnth* l *0*)) (*nfusecat* (*nsubn-nellist* σ l)) =
ndropn (*nfirst* (*NCons* x l)) σ
proof −
assume *a*: *nlength* l > *0*
show *nfuse* (*nsubn* σ x (*nnth* l *0*)) (*nfusecat* (*nsubn-nellist* σ l)) =
ndropn (*nfirst* (*NCons* x l)) σ
proof −
have *50*: ∃ *y l1*. l = (*NCons y l1*)
using *a nellist-split-2-first* **by** *auto*
have *51*: (*nfusecat* (*nsubn-nellist* σ l)) = *ndropn* (*nfirst* l) σ
using *NCons.IH*[*of* σ] *NCons.prems* **by** *auto*
have *52*: *nlast* (*nsubn* σ x (*nnth* l *0*)) = *nfirst* (*ndropn* (*nfirst* l) σ)
by (*metis NCons.prems*(*2*) *diff-add nfirst-eq-nnth-zero nidx-LCons-1 nnth-zero-ndropn nsubn-nlast order.strict-iff-order*)
have *53*: *enat* x ≤ *enat* (*nnth* l *0*)
by (*metis NCons.prems*(*2*) *enat-ord-simps*(*1*) *nfirst-eq-nnth-zero nidx-LCons-1 order.strict-iff-order*)
have *54*: *enat* (*nnth* l *0*) ≤ *enat* (*the-enat* (*nlength* σ))
by (*metis NCons.hyps NCons.prems*(*2,3*) *a enat-ord-simps*(*1*) *nidx-LCons-1 nidx-all-le-nlast nlast-NCons order.strict-iff-order zero-enat-def*)
have *55*: *enat* (*the-enat* (*nlength* σ)) ≤ *nlength* σ
by (*metis dual-order.refl enat-ord-simps*(*3*) *enat-the-enat*)
have *56* : *nfuse* (*nsubn* σ x (*nnth* l *0*)) (*nsubn* σ (*nnth* l *0*) (*the-enat* (*nlength* σ))) = *nsubn* σ x (*the-enat* (*nlength* σ))
using *nsubn-nfuse*[*of* x (*nnth* l *0*) *the-enat*(*nlength* σ) σ]
using *53 54 55* **by** *auto*
have *57*: (*ndropn* (*nfirst* l) σ) = (*nsubn* σ (*nnth* l *0*) (*the-enat* (*nlength* σ)))
by (*metis NCons.prems*(*1*) *idiff-enat-enat ndropn-nlength nfinite-conv-nlength-enat nfirst-eq-nnth-zero nle-le*

nsubn-def1 ntaken-all the-enat.simps)
have *58*: *ndropn* (*nfirst* (*NCons x l*)) *σ* = (*ndropn x σ*)
by (*simp add*: *nfirst-eq-nnth-zero*)
have *59*: *nfuse* (*nsubn σ x* (*nnth l 0*)) (*ndropn* (*nfirst l*) *σ*) =
nfuse (*nsubn σ x* (*nnth l 0*)) (*nsubn σ* (*nnth l 0*) (*the-enat* (*nlength σ*)))
by (*simp add*: *57*)
have *60*: *nsubn σ x* (*the-enat* (*nlength σ*)) = (*ndropn x σ*)
by (*metis NCons.prems*(*1*) *idiff-enat-enat ndropn-nlength nfinite-conv-nlength-enat nle-le nsubn-def1*
ntaken-all
the-enat.simps)
show *?thesis*
by (*simp add*: *51 56 57 58 60*)
qed
qed
show *?thesis*
by (*simp add*: *2 4 5*)
qed
qed

lemma *nfusecat-nsubn-nellist*:
assumes *nidx l*
¬*nfinite σ*
¬*nfinite l*
shows *nfusecat* (*nsubn-nellist σ l*) = *ndropn* (*nfirst l*) *σ*
using *assms*
proof (*coinduction arbitrary*: *l σ*)
case (*Eq-nellist l1 σ1*)
then show *?case*
proof −
have *1*: *nidx l1* ⟹
¬ *nfinite σ1* ⟹
¬ *nfinite l1* ⟹
is-NNil (*nfusecat* (*nsubn-nellist σ1 l1*)) = *is-NNil* (*ndropn* (*nfirst l1*) *σ1*)
using *nnth-nsubn-nellist*[*of* - *l1 σ1*] **unfolding** *nidx-expand* **apply** *auto*
apply (*metis enat-0-iff*(*1*) *is-NNil-nfusecat-b nellist.disc*(*2*) *nellist-split-2-first nfinite-ntaken nle-le*
nsubn-nlength-gr-one
ntaken-all zero-le)
using *is-NNil-imp-nfinite nfinite-ndropn-b* **by** *blast*
have *2*: *nidx l1* ⟹
¬ *nfinite σ1* ⟹
¬ *nfinite l1* ⟹
is-NNil (*nfusecat* (*nsubn-nellist σ1 l1*)) = *is-NNil* (*ndropn* (*nfirst l1*) *σ1*) ∧
(*is-NNil* (*nfusecat* (*nsubn-nellist σ1 l1*)) ⟶
is-NNil (*ndropn* (*nfirst l1*) *σ1*) ⟶ *nlast* (*nfusecat* (*nsubn-nellist σ1 l1*)) = *nlast* (*ndropn* (*nfirst l1*)
σ1)) ∧
(¬ *is-NNil* (*nfusecat* (*nsubn-nellist σ1 l1*)) ⟶
¬ *is-NNil* (*ndropn* (*nfirst l1*) *σ1*) ⟶
nhd (*nfusecat* (*nsubn-nellist σ1 l1*)) = *nhd* (*ndropn* (*nfirst l1*) *σ1*))
using *nnth-nsubn-nellist*[*of* - *l1 σ1*] **unfolding** *nidx-expand* **apply** *auto*

using *1 Eq-nellist(1)* **apply** *blast*
using *1 Eq-nellist(1)* **apply** *blast*
using *is-NNil-imp-nfinite nfinite-ndropn* **apply** *blast*
by (*metis enat-ord-simps(3) ndropn-nfirst nfinite-conv-nlength-enat nfirst-eq-nnth-zero nfirst-nfusecat-nfirst nhd-conv-nnth not-infinity-eq nsubn-nfirst*)
have *3*: *nidx l1* $\Longrightarrow$
$\neg$ *nfinite σ1* $\Longrightarrow$
$\neg$ *nfinite l1* $\Longrightarrow$ ($\exists$ *l σ*. *ntl* (*nfusecat* (*nsubn-nellist σ1 l1*)) = *nfusecat* (*nsubn-nellist σ l*) $\wedge$ *ntl* (*ndropn* (*nfirst l1*) *σ1*) = *ndropn* (*nfirst l*) *σ* $\wedge$ *nidx l* $\wedge$ $\neg$ *nfinite σ* $\wedge$ $\neg$ *nfinite l*)
proof −
assume *a0*: *nidx l1*
assume *a1*: ¬*nfinite σ1*
assume *a2*: ¬*nfinite l1*
show ($\exists$ *l σ*. *ntl* (*nfusecat* (*nsubn-nellist σ1 l1*)) = *nfusecat* (*nsubn-nellist σ l*) $\wedge$ *ntl* (*ndropn* (*nfirst l1*) *σ1*) = *ndropn* (*nfirst l*) *σ* $\wedge$ *nidx l* $\wedge$ $\neg$ *nfinite σ* $\wedge$ $\neg$ *nfinite l*)
proof −
have *4*: *ntl* (*ndropn* (*nfirst l1*) *σ1*) = (*ndropn* (*Suc* (*nfirst l1*)) *σ1*)
by (*metis ndropn-Suc-NCons nellist.sel(5) ntl-ndropn*)
have *5*: *ntl*(*nfusecat* (*nsubn-nellist σ1 l1*)) =
(*if is-NNil* (*nsubn-nellist σ1 l1*) *then ntl* (*nfirst* (*nsubn-nellist σ1 l1*)) *else*
(*if is-NNil* (*nfirst* (*nsubn-nellist σ1 l1*)) *then*
if is-NNil (*nfusecat* (*ntl* (*nsubn-nellist σ1 l1*)))
then ntl (*nfirst* (*nsubn-nellist σ1 l1*))
else ntl (*nfusecat* (*ntl* (*nsubn-nellist σ1 l1*)))
else
if is-NNil (*nfusecat* (*ntl* (*nsubn-nellist σ1 l1*)))
then ntl (*nfirst* (*nsubn-nellist σ1 l1*))
else nappend (*ntl* (*nfirst* (*nsubn-nellist σ1 l1*))) (*ntl* (*nfusecat* (*ntl* (*nsubn-nellist σ1 l1*))))))
by (*simp add*: *ntl-lfusecat*)
have *6*: $\neg$ *is-NNil* (*nsubn-nellist σ1 l1*)
by (*metis a2 nellist.disc(2) nellist-split-2-first nfinite-ntl nlength-eq-enat-nfiniteD nlength-nsubn-nellist not-gr-zero zero-enat-def*)
have *7*: $\neg$ *is-NNil* (*nfirst* (*nsubn-nellist σ1 l1*))
by (*metis Eq-nellist(1) a1 a2 is-NNil-ndropn linorder-not-less ndropn-0 nfinite-ndropn nfirst-eq-nnth-zero nidx-expand nle-le nlength-eq-enat-nfiniteD nnth-nsubn-nellist not-gr-zero nsubn-nlength-gr-one zero-enat-def*)
have *8*: ¬*nfinite* (*ntl* (*nsubn-nellist σ1 l1*))
by (*metis a2 nfinite-conv-nlength-enat nfinite-ntl nlength-nsubn-nellist not-gr-zero zero-enat-def*)
have *9*: $\forall$ *nell*$\in$*nset* (*ntl* (*nsubn-nellist σ1 l1*)). *0* < *nlength nell*
using *nsubn-nellist-ndropn*[*of l1 1 σ1*] *nnth-nsubn-nellist*[*of - l1 σ1*] **using** *a0 nidx-expand* **apply** *auto*
by (*metis a1 a2 in-nset-conv-nnth nfinite-ntaken nle-le nnth-ntl not-gr-zero nsubn-nlength-gr-one ntaken-all*)
have *10*: $\forall$ *nell*$\in$*nset* (*ntl* (*nsubn-nellist σ1 l1*)). *nfinite nell*
using *nsubn-nellist-ndropn*[*of l1 1 σ1*] *nnth-nsubn-nellist*[*of - l1 σ1*] **using** *a0 nidx-expand* **apply** *auto*
by (*metis a2 enat-ord-simps(3) in-nset-conv-nnth in-nset-ntlD nfinite-conv-nlength-enat not-infinity-eq nsubn-nfinite*)

have *11*: ¬*nfinite* (*nfusecat* (*ntl* (*nsubn-nellist* σ *1 l1*)))
using *nfusecat-nfinite*[*of* (*ntl* (*nsubn-nellist* σ *1 l1*))]
by (*metis 10 6 8 9 a0 a1 a2 nellist.collapse*(*2*)
nlastnfirst-LCons
nlastnfirst-nsubn-nellist-infinite)
have *12*: ¬ *is-NNil* (*nfusecat* (*ntl* (*nsubn-nellist* σ *1 l1*)))
using *11* **by** *auto*
have *13*: *ntl*(*nfusecat* (*nsubn-nellist* σ *1 l1*)) = *nappend* (*ntl* (*nfirst* (*nsubn-nellist* σ *1 l1*))) (*ntl* (*nfusecat* (*ntl* (*nsubn-nellist* σ *1 l1*))))
using *5* **by** (*simp add*: *12 6 7*)

have *14*: ∃ *lx l11*. *l1*= (*NCons lx l11*)
by (*metis a2 is-NNil-nappend nappend-inf nellist.collapse*(*2*))
have *15*: ∃ *lns ln*. *nsubn-nellist* σ *1 l1* = (*NCons ln* (*nsubn-nellist* σ *1 lns*))
by (*metis 14 a0 a1 a2 nfinite-NCons nidx-LCons-1 nsubn-nellist-ncons*)
have *150*: *nfirst* (*nsubn-nellist* σ *1 l1*) = (*nsubn* σ *1* (*nnth l1 0*) (*nnth l1* (*Suc 0*)))
by (*metis Suc-ile-eq a2 nfinite-ntaken nfirst-eq-nnth-zero nnth-nsubn-nellist not-gr-zero ntaken-all zero-enat-def zero-le*)
have *141*: *nlength*(*nfirst*((*nsubn-nellist* σ *1 l1*))) = (*nnth l1* (*Suc 0*)) − (*nnth l1 0*)
by (*metis 150 a1 idiff-enat-enat min.cobounded1 min-def ndropn-nlength nfinite-ndropn nsubn-def1 nsubn-nfinite nsubn-nlength ntaken-all*)
have *16*: (*nfusecat* (*nsubn-nellist* σ *1 l1*)) =
nfuse (*nfirst*((*nsubn-nellist* σ *1 l1*))) (*nfusecat* (*ntl*((*nsubn-nellist* σ *1 l1*))))
by (*simp add*: *6 nfusecat-expand*)
have *17*: *nlength*(*nfirst*((*nsubn-nellist* σ *1 l1*))) > *1* ⟹
ntl (*nfusecat* (*nsubn-nellist* σ *1 l1*)) = *nfuse* (*ntl*(*nfirst*((*nsubn-nellist* σ *1 l1*)))) (*nfusecat* (*ntl*((*nsubn-nellist* σ *1 l1*))))
by (*simp add*: *5 6 7 nfuse-def1*)
have *18*: *nlength*(*nfirst*((*nsubn-nellist* σ *1 l1*))) > *1* ⟹
nfuse (*ntl*(*nfirst*((*nsubn-nellist* σ *1 l1*)))) (*nfusecat* (*ntl*((*nsubn-nellist* σ *1 l1*)))) =
nfusecat ((*NCons* (*ntl*(*nfirst*((*nsubn-nellist* σ *1 l1*)))) (*ntl*((*nsubn-nellist* σ *1 l1*)))))
by *simp*
have *19*: (*ntl*(*nfirst*((*nsubn-nellist* σ *1 l1*)))) =
ntl(*nsubn* σ *1* (*nnth l1 0*) (*nnth l1 1*))
by (*metis 14 One-nat-def a0 a1 a2 nellist.sel*(*5*) *nfinite-NCons nfirst-eq-nnth-zero nidx-LCons-1 nnth-0 nnth-ntl*
nsubn-nellist-ncons)
have *20*: *nlength*(*nfirst*((*nsubn-nellist* σ *1 l1*))) > *1* ⟹
ntl(*nsubn* σ *1* (*nnth l1 0*) (*nnth l1 1*)) = *nsubn* σ *1* (*Suc* (*nnth l1 0*)) (*nnth l1 1*)
using *141 14 4 a0* **apply** *simp*
by (*metis add-diff-cancel-right diff-add nfirst-eq-nnth-zero nidx-LCons-1 nnth-0 nnth-Suc-NCons not-less-zero nsubn-def1*
ntl-ntaken order-less-le plus-1-eq-Suc zero-enat-def)
have *21*: *nlength*(*nfirst*((*nsubn-nellist* σ *1 l1*))) > *1* ⟹
(*NCons* (*ntl*(*nfirst*((*nsubn-nellist* σ *1 l1*)))) (*ntl*((*nsubn-nellist* σ *1 l1*)))) =
nsubn-nellist σ *1* (*NCons* (*Suc* (*nnth l1 0*)) (*ntl l1*))
using *20 14 7 19 a0 a1 a2* **apply** *simp*
by (*metis Suc-lessI nellist.collapse*(*2*) *nellist.sel*(*5*) *nfinite-ntl nfirst-eq-nnth-zero nidx-LCons-1 nlength-NCons nlength-NNil nnth-0*
nnth-Suc-NCons nsubn-nellist-ncons nsubn-same one-eSuc order.strict-iff-order)

have *22*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) > *1* $\Longrightarrow$
ntl (*nfusecat* (*nsubn-nellist* σ*1 l1*)) = *nfusecat* (*nsubn-nellist* σ*1* (*NCons* (*Suc* (*nnth l1 0*)) (*ntl l1*)))
by (*simp add*: *17 18 21*)
have *23*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) > *1* $\Longrightarrow$
ntl (*ndropn* (*nfirst l1*) σ*1*) = *ndropn* (*nfirst* ((*NCons* (*Suc* (*nnth l1 0*)) (*ntl l1*)))) σ*1*
by (*metis 4 nfirst-eq-nnth-zero nnth-0*)
have *24*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) > *1* $\Longrightarrow$
nidx ((*NCons* (*Suc* (*nnth l1 0*)) (*ntl l1*)))
by (*metis 14 Suc-lessI a0 a1 a2 diff-add-inverse2 idiff-enat-enat less-numeral-extra*(*4*) *min-less-iff-conj nellist.sel*(*5*) *nfinite-ntl*
nfirst-eq-nnth-zero nidx-LCons-1 nnth-0 nsubn-nellist-ncons nsubn-nlength one-enat-def plus-1-eq-Suc)

have *25*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) > *1* $\Longrightarrow$
$\neg$ *nfinite* ((*NCons* (*Suc* (*nnth l1 0*)) (*ntl l1*)))
by (*simp add*: *a2*)
have *26*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) > *1* $\Longrightarrow$ *?thesis*
using *22 23 24 25 a1* **by** *blast*
have *27*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) = *1* $\Longrightarrow$
(*ntl*(*nfirst*((*nsubn-nellist* σ*1 l1*)))) = *nsubn* σ*1* (*nnth l1* (*Suc 0*)) (*nnth l1* (*Suc 0*))
by (*metis 141 19 Suc-eq-plus1 add.commute add-diff-inverse-nat diff-is-0-eq enat-1-iff*(*2*) *epred-1 epred-conv-minus le-add2 ndropn-0*
ndropn-Suc-NCons ndropn-nlast ndropn-nlength nellist.sel(*5*) *nfinite-ntl nlast-ntl nlength-eq-enat-nfiniteD nsubn-nlast nsubn-same*
ntl-ndropn order.strict-iff-order the-enat-0 zero-neq-one)
have *271*: *nsubn* σ*1* (*nnth l1* (*Suc 0*)) (*nnth l1* (*Suc 0*)) = (*NNil* (*nnth* σ*1* (*nnth l1* (*Suc 0*))))
using *nsubn-same* **by** *blast*
have *272*: (*ntl*((*nsubn-nellist* σ*1 l1*))) = (*NCons* (*nsubn* σ*1* (*nnth l1* (*Suc 0*)) (*nnth l1* (*Suc* (*Suc 0*)))) (*ntl*(*ntl*(*nsubn-nellist* σ*1 l1*))))

by (*metis 8 Suc-ile-eq a2 dual-order.strict-iff-order i0-iless-eSuc is-NNil-imp-nfinite nellist.collapse*(*2*) *nlength-NCons*
nlength-eq-enat-nfiniteD nnth-0 nnth-nsubn-nellist nnth-ntl zero-enat-def)
have *272*: *nfirst* (*nfusecat* (*ntl*((*nsubn-nellist* σ*1 l1*)))) =
nfirst (*nsubn* σ*1* (*nnth l1* (*Suc 0*)) (*nnth l1* (*Suc 0*)))
using *nfirst-nfusecat*[*of*] *272* **by** (*metis nsubn-nfirst*)
have *28*: *nfuse* (*nsubn* σ*1* (*nnth l1* (*Suc 0*)) (*nnth l1* (*Suc 0*))) (*nfusecat* (*ntl*((*nsubn-nellist* σ*1 l1*)))) =
(*nfusecat* (*ntl*((*nsubn-nellist* σ*1 l1*))))
using *14 15 19 a0 a1 a2 272*
by (*metis nfuse-leftneutral nsubn-nfirst nsubn-same*)
have *29*: (*nfusecat* (*ntl*((*nsubn-nellist* σ*1 l1*)))) =
(*nfusecat* (((*nsubn-nellist* σ*1* (*ntl l1*)))))
using *14 a0 a1 a2 nsubn-nellist-ncons* **by** *fastforce*
have *30*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) = *1* $\Longrightarrow$
ntl (*nfusecat* (*nsubn-nellist* σ*1 l1*)) = (*nfusecat* (((*nsubn-nellist* σ*1* (*ntl l1*)))))
by (*metis 27 28 29 5 6 7 nfuse-def1*)
have *31*: *nlength*(*nfirst*((*nsubn-nellist* σ*1 l1*))) = *1* $\Longrightarrow$
ntl (*ndropn* (*nfirst l1*) σ*1*) = *ndropn* (*nfirst* (*ntl l1*)) σ*1*
using *14 141 4* **apply** *simp*
by (*metis a0 add.commute add-diff-inverse-nat enat.inject less-numeral-extra*(*3*) *nat-diff-split nel-*

list.sel(5) nfirst-eq-nnth-zero
nidx-LCons-1 nnth-0 nnth-Suc-NCons one-enat-def plus-1-eq-Suc zero-less-diff)
have *32*: *nlength*(*nfirst*((*nsubn-nellist σ1 l1*))) = *1* ⟹
nidx (*ntl l1*)
using *14 a0* **by** *auto*
have *33*: *nlength*(*nfirst*((*nsubn-nellist σ1 l1*))) = *1* ⟹
¬ *nfinite* (*ntl l1*)
by (*simp add*: *a2*)
have *34*: *nlength*(*nfirst*((*nsubn-nellist σ1 l1*))) = *1* ⟹ *?thesis*
using *30 31 32 33 a1* **by** *blast*
have *35*: *nlength*(*nfirst*((*nsubn-nellist σ1 l1*))) = *1* ∨ *nlength*(*nfirst*((*nsubn-nellist σ1 l1*))) > *1*
by (*metis 7 dual-order.order-iff-strict i0-lb ileI1*
is-NNil-ndropn ndropn-0 one-eSuc)
show *?thesis*
using *26 34 35* **by** *blast*
qed
qed
show *?thesis*
using *2 3 Eq-nellist*(*1*,*2*,*3*) **by** *blast*
qed
qed

lemma *nfusecat-nsubn-nellist-finite-var*:
assumes *nfinite l*
nfirst l =*0*
nfinite σ
nidx l
nlast l = (*the-enat*(*nlength σ*))
shows *nfusecat* (*nsubn-nellist σ l*) = *σ*
using *assms nfusecat-nsubn-nellist-finite* **by** (*metis ndropn-0*)

lemma *nfusecat-nsubn-nellist-var*:
assumes *nidx l*
nfirst l =*0*
¬*nfinite σ*
¬*nfinite l*
shows *nfusecat* (*nsubn-nellist σ l*) = *σ*
using *assms nfusecat-nsubn-nellist* **by** (*metis ndropn-0*)

lemma *nfilter-nsubn-nlength-gr-zero-finite*:
assumes *nidx l*
nfirst l =*0*
nfinite σ
nfinite l
nlast l = (*the-enat*(*nlength σ*))
⋀*i*. *i* < *nlength l* ⟹ *P* (*nfirst* (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))
⋀*i*. *i* < *nlength l* ⟹ *P* (*nlast* (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))
i < *nlength l*
shows *nlength* (*nfilter P* (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)))) >*0*
proof −

have *1*: $P\ (nnth\ \sigma\ (nnth\ l\ i))$
using *assms* **by** (*metis nsubn-nfirst*)
have *2*: $P\ (nnth\ \sigma\ (nnth\ l\ (Suc\ i)))$
using *assms*
by (*simp add*: *Suc-ile-eq dual-order.strict-iff-order nidx-expand nsubn-nlast*)
have *3*: $nfirst\ (nfilter\ P\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) = (nnth\ \sigma\ (nnth\ l\ i))$
using *assms*
by (*metis nfilter-NNil nfilter-chop1-ntaken nfirst-eq-nnth-zero nlength-NNil nsubn-nfirst ntaken-0 ntaken-nlast the-enat-0 zero-enat-def zero-le*)
have *30* : $enat\ (nnth\ l\ (Suc\ i) - nnth\ l\ i) \leq nlength\ (ndropn\ (nnth\ l\ i)\ \sigma)$
using *assms*
by (*metis eSuc-enat enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat ileI1 ndropn-nlength nfinite-conv-nlength-enat nidx-all-le-nlast the-enat.simps*)
have *4*: $nlast\ (nfilter\ P\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) = (nnth\ \sigma\ (nnth\ l\ (Suc\ i)))$
unfolding *nsubn-def1* **using** *assms nfilter-nlast*[*of* $(nnth\ l\ (Suc\ i) - nnth\ l\ i)\ (ndropn\ (nnth\ l\ i)\ \sigma)$]
by (*metis 30 diff-add dual-order.strict-iff-order eSuc-enat ileI1 nidx-expand nsubn-def1 nsubn-nlast*)
have *5*: $nlength\ (nkfilter\ P\ 0\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) = nlength\ (nfilter\ P\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
using *assms* **by** (*meson nkfilter-nlength nset-nlast nsubn-nfinite*)
have *6*: $nfinite\ (nfilter\ P\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
by (*metis length-nfilter-le nfinite-nlength-enat nfinite-ntaken nsubn-def1 ntaken-all*)
have *60*: $\exists x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ P\ x$
using *assms nset-nlast nsubn-nfinite* **by** *blast*
have *61*: $\exists x{\in}nset\ (ntaken\ (nnth\ l\ (Suc\ i) - nnth\ l\ i)\ (ndropn\ (nnth\ l\ i)\ \sigma))\ .\ P\ x$
using *assms*
by (*metis nset-nlast nsubn-def1 nsubn-nfinite*)
have *62*: $enat\ (nnth\ l\ (Suc\ i) - nnth\ l\ i) \leq nlength\ (ndropn\ (nnth\ l\ i)\ \sigma)$
using *assms* **using** *30* **by** *blast*
have *63*: $(nnth\ l\ (Suc\ i) - nnth\ l\ i) > 0$
using *assms*
by (*simp add*: *Suc-ile-eq nidx-expand*)
have *7*: $nlength\ (nkfilter\ P\ 0\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) > 0$
using *assms nkfilter-nlength*[*of* $(nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ P\ 0$]
unfolding *nsubn-def1*
using *nkfilter-nappend-ntaken*[*of* $(nnth\ l\ (Suc\ i) - nnth\ l\ i)\ (ndropn\ (nnth\ l\ i)\ \sigma)\ P\ 0$]
by (*metis 1 61 62 63 le-add-same-cancel2 le-zero-eq linorder-not-less ndropn-nfirst nfirst-eq-nnth-zero nkfilter-nfirst nkfilter-nlast ntaken-nlast the-enat-0*)
show *?thesis*
using *5 7* **by** *auto*
qed

lemma *nfilter-nsubn-nlength-gr-zero*:
assumes *nidx l*
$nfirst\ l = 0$
$\neg\ nfinite\ \sigma$
$\neg\ nfinite\ l$
$\bigwedge i.\ P\ (nfirst\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
$\bigwedge i.\ P\ (nlast\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$

shows *nlength (nfilter P (nsubn σ (nnth l i) (nnth l (Suc i)))) >0*
proof −
have *1*: *P (nnth σ (nnth l i))*
by (*metis assms(5) nsubn-nfirst*)
have *2*: *P (nnth σ (nnth l (Suc i)))*
by (*metis assms(5) nsubn-nfirst*)
have *3*: *nfirst (nfilter P (nsubn σ (nnth l i) (nnth l (Suc i)))) = (nnth σ (nnth l i))*
by (*metis assms(5) ndropn-0 nfilter-nfirst nsubn-nfirst*)
have *4*: *nlast (nfilter P (nsubn σ (nnth l i) (nnth l (Suc i)))) = (nnth σ (nnth l (Suc i)))*
unfolding *nsubn-def1* **using** *nfilter-nlast*[*of*]
by (*metis assms(1,3,4,6) diff-add dual-order.strict-iff-order nfinite-ndropn nfinite-ntaken nidx-expand nle-le nsubn-def1 nsubn-nlast ntaken-all*)
have *5*: *nlength (nkfilter P 0 (nsubn σ (nnth l i) (nnth l (Suc i)))) = nlength (nfilter P (nsubn σ (nnth l i) (nnth l (Suc i))))*
by (*meson assms(6) nkfilter-nlength nset-nlast nsubn-nfinite*)
have *6*: *nfinite (nfilter P (nsubn σ (nnth l i) (nnth l (Suc i))))*
by (*metis length-nfilter-le nfinite-nlength-enat nfinite-ntaken nsubn-def1 ntaken-all*)
have *60*: ∃ *x*∈*nset (nsubn σ (nnth l i) (nnth l (Suc i))). P x*
using *assms nset-nlast nsubn-nfinite* **by** *blast*
have *61*: ∃ *x*∈*nset (ntaken (nnth l (Suc i) − nnth l i) (ndropn (nnth l i) σ)). P x*
using *assms*
by (*metis nset-nlast nsubn-def1 nsubn-nfinite*)
have *62*: *enat (nnth l (Suc i) − nnth l i) ≤ nlength (ndropn (nnth l i) σ)*
using *assms*
by (*metis nfinite-ndropn nle-le nsubn-def1 nsubn-nfinite ntaken-all*)
have *63*: *(nnth l (Suc i) − nnth l i) > 0*
using *assms*
by (*metis nfinite-ntaken nidx-expand nle-le ntaken-all zero-less-diff*)
have *7*: *nlength (nkfilter P 0 (nsubn σ (nnth l i) (nnth l (Suc i)))) > 0*
using *assms nkfilter-nlength*[*of (nsubn σ (nnth l i) (nnth l (Suc i))) P 0*]
unfolding *nsubn-def1*
using *nkfilter-nappend-ntaken*[*of (nnth l (Suc i) − nnth l i) (ndropn (nnth l i) σ) P 0*]
by (*metis 1 61 62 63 le-add-same-cancel2 le-zero-eq linorder-not-less ndropn-nfirst nfirst-eq-nnth-zero nkfilter-nfirst nkfilter-nlast ntaken-nlast the-enat-0*)
show *?thesis*
using *5 7* **by** *auto*
qed

lemma *nlength-nmap-nsubn-nellist-gr-zero-finite*:
assumes *nidx l*
nfirst l =0
nfinite σ
nfinite l
nlast l = (the-enat(nlength σ))
⋀*i. i < nlength l* ⟹ *P (nfirst (nnth (nsubn-nellist σ l) i))*
⋀*i. i < nlength l* ⟹ *P (nlast (nnth (nsubn-nellist σ l) i))*
i < nlength l
shows *nlength (nnth (nmap (nfilter P) (nsubn-nellist σ l)) i) >0*
proof −
have *1*: *(nnth (nmap (nfilter P) (nsubn-nellist σ l)) i) = nfilter P (nnth (nsubn-nellist σ l) i)*

using *assms nnth-nmap*[*of i* (*nsubn-nellist* σ *l*) (*nfilter P*)]
by (*metis enat-ile nfinite-conv-nlength-enat nlast-nmap nle-le nlength-nmap ntaken-all ntaken-nlast*)
have *2*: *nfilter P* (*nnth* (*nsubn-nellist* σ *l*) *i*) =
nfilter P (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))
using *nnth-nsubn-nellist*[*of i l* σ] *assms Suc-ile-eq* **by** *presburger*
have *3*: *nlength* (*nfilter P* (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))) > *0*
using *assms nfilter-nsubn-nlength-gr-zero*[*of l* σ *P*]
by (*metis Suc-ile-eq nfilter-nsubn-nlength-gr-zero-finite nnth-nsubn-nellist*)
show *?thesis*
using *1 2 3* **by** *auto*
qed

lemma *nlength-nmap-nsubn-nellist-gr-zero*:
assumes *nidx l*
nfirst l =*0*
¬ *nfinite* σ
¬ *nfinite l*
$\bigwedge$*i*. *P* (*nfirst* (*nnth* (*nsubn-nellist* σ *l*) *i*))
$\bigwedge$*i*. *P* (*nlast* (*nnth* (*nsubn-nellist* σ *l*) *i*))
shows *nlength* (*nnth* (*nmap* (*nfilter P*) (*nsubn-nellist* σ *l*)) *i*) >*0*
proof −
have *1*: (*nnth* (*nmap* (*nfilter P*) (*nsubn-nellist* σ *l*)) *i*) = *nfilter P* (*nnth* (*nsubn-nellist* σ *l*) *i*)
using *assms nnth-nmap*[*of i* (*nsubn-nellist* σ *l*) (*nfilter P*)]
by (*metis enat-ile nfinite-conv-nlength-enat nlast-nmap nle-le nlength-nmap ntaken-all ntaken-nlast*)
have *2*: *nfilter P* (*nnth* (*nsubn-nellist* σ *l*) *i*) =
nfilter P (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))
using *nnth-nsubn-nellist*[*of i l* σ] *assms*
by (*metis linorder-le-cases nfinite-ntaken ntaken-all*)
have *3*: *nlength* (*nfilter P* (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))) > *0*
using *assms nfilter-nsubn-nlength-gr-zero*[*of l* σ *P*]
by (*metis nfinite-ntaken nle-le nnth-nsubn-nellist not-gr-zero ntaken-all*)
show *?thesis*
using *1 2 3* **by** *auto*
qed

lemma *nfusecat-nconcat-nmap-nbutlast-zero*:
assumes $\bigwedge$ *xx*. *xx* ∈ *nset ll* ⟹ *nlength xx* >*0*
$\bigwedge$ *xx*. *xx* ∈ *nset ll* ⟹ *nfinite xx*
nlastnfirst ll
nfinite ll
nlength ll = *0*
shows *nfusecat ll* = *nfirst ll*
using *assms*
by (*metis NNil-eq-ntake-iff nfusecat-NNil nle-le ntake-all*)

lemma *nfusecat-nconcat-nmap-nbutlast-finite*:
assumes $\bigwedge$ *xx*. *xx* ∈ *nset ll* ⟹ *nlength xx* >*0*
$\bigwedge$ *xx*. *xx* ∈ *nset ll* ⟹ *nfinite xx*

```
        nlastnfirst ll
        nfinite ll
        nlength ll >0
 shows   nfusecat ll = (   nappend (nconcat (nbutlast (nmap (nbutlast) ll))) (nlast ll))
proof −
 let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
 have 1: nfusecat ll = (nellist-of-llist ∘ lfusecat ∘ ?n2l) ll
   unfolding nfusecat-def2 by simp
 have 01: (nellist-of-llist ∘ lfusecat ∘ ?n2l) ll =
         nellist-of-llist (lfusecat (lmap llist-of-nellist (llist-of-nellist ll)))
    by simp
 have 2: nconcat (nmap (nbutlast) ll) =
        (nellist-of-llist ∘ lconcat ∘ ?n2l) (nmap nbutlast ll)
   unfolding nconcat-def2 by simp
 have 3: ⋀xx. xx ∈ nset ll ⟹(llist-of-nellist (nbutlast xx)) = lbutlast (llist-of-nellist xx)
       unfolding nbutlast-def apply simp
     by (metis assms(1) lbutlast.ctr(1) less-numeral-extra(3) llength-LNil llength-lbutlast llength-llist-of-nellist)
 have 4: lfinite (llist-of-nellist ll)
   using assms(4) nfinite-def by auto
 have 7: (⋀xx. xx ∈ lset (lmap llist-of-nellist (llist-of-nellist ll)) ⟹ 1 < llength xx)
    using assms apply auto
    by (metis epred-1 llength-llist-of-nellist llist-of-nellist.simps(1) nless-le not-lnull-llength)
 have 8: (⋀xx. xx ∈ lset (lmap llist-of-nellist (llist-of-nellist ll)) ⟹ lfinite xx)
    using assms apply auto    using nfinite-def by blast
 have 9: llastlfirst (lmap llist-of-nellist (llist-of-nellist ll))
   using assms nlastnfirst.rep-eq by auto
 have 10:  lfinite (lmap llist-of-nellist (llist-of-nellist ll))
    using assms
    using 4 lfinite-lmap by blast

 have 40: (nellist-of-llist ∘ lconcat ∘ ?n2l) (nmap nbutlast ll) =
    nellist-of-llist (lconcat (lmap llist-of-nellist (lmap nbutlast (llist-of-nellist ll))))
   by simp
 have 41: ( (lmap (lbutlast∘llist-of-nellist) (llist-of-nellist ll))) =
         ( (lmap lbutlast (lmap llist-of-nellist (llist-of-nellist ll) )))
  by (simp add: llist.map-comp)
 have 411: ( (lmap llist-of-nellist (lmap nbutlast (llist-of-nellist ll)))) =
         ( (lmap (llist-of-nellist∘ nbutlast) (llist-of-nellist ll)))
    by (simp add: llist.map-comp)
 have 42: ⋀xx. xx∈ lset (llist-of-nellist ll) ⟹ 0< nlength xx
     using assms(1) by auto
 have 420: (⋀z. z ∈ lset (llist-of-nellist ll) ⟹ (llist-of-nellist ∘ nbutlast) z = (lbutlast ∘ llist-of-nellist) z)
    using 3 by auto
 have 43: ( (lmap (llist-of-nellist∘  nbutlast) (llist-of-nellist ll))) =
         ( (lmap (lbutlast∘ llist-of-nellist) (llist-of-nellist ll)))
    using 3 4 assms lbutlast-not-lfinite[of (lmap llist-of-nellist (llist-of-nellist ll))]
using lset-llist-of-nellist-a[of ll] using  comp-apply llist.map-cong0[of (llist-of-nellist ll)
    (llist-of-nellist∘  nbutlast) (lbutlast∘ llist-of-nellist)   ]
by auto
 have 6: (nellist-of-llist ∘ lconcat ∘ ?n2l) (nmap nbutlast ll) =
```

nellist-of-llist (*lconcat* (*lmap* (*lbutlast*∘*llist-of-nellist*) (*llist-of-nellist ll*)))
using *40 411 43* **by** *presburger*
have *11*: (*lfusecat* (*lmap llist-of-nellist* (*llist-of-nellist ll*))) =
lappend (*lconcat* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))))
(*llast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))
using *lfusecat-lconcat-lmap-lbutlast-lfinite*[*of* (*lmap llist-of-nellist* (*llist-of-nellist ll*))]
using *10 7 8 9* **by** *fastforce*
have *12*: *nellist-of-llist* (*lfusecat* (*lmap llist-of-nellist* (*llist-of-nellist ll*))) =
nellist-of-llist (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))))
(*llast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))
using *11* **by** *simp*

have *122*: ⋀*z*. *z* ∈ *lset* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))) ⟹ *llength z* > *0*
using *41 411 43*
by (*metis i0-less in-lset-conv-lnth llength-eq-0 llength-lmap llist-of-nellist-not-lnull lnth-lmap*)
have *123*: *llength* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))) > *0*
using *assms*(*5*) **by** *auto*
have *124*: ⋀*z*. *z* ∈ *lset* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))) ⟹ *llength z* > *0*
using *122* **by** (*meson in-lset-lbutlastD*)
have *125*: *lfinite* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))
using *10 lbutlast-lfinite lfinite-lmap* **by** *blast*
have *126*: ¬ *lnull* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))
using *123* **by** (*metis i0-less llength-eq-0*)
have *127*: *lconcat* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))) =
(*if lnull* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))) *then LNil*
else lappend (*lconcat* (*lbutlast* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))))
(*llast* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))))
using *lconcat-split-llast*[*of* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))]
using *125* **by** *blast*
have *128*: *lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))) =
lmap lbutlast (*lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))
using *lmap-lbutlast*[*of lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))]
by *simp*
have *129*: (*llast* (*lmap lbutlast* (*lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))) =
lbutlast (*llast* (*lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))
using *llast-lmap*[*of* (*lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))) *lbutlast*]
using *125 126 128 lfinite-lmap llist.map-disc-iff* **by** *auto*
have *130*: *llength* (*lbutlast* (*llast* (*lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))) > *0*
using *124 125 126 128 129*
by (*metis* (*no-types*, *lifting*) *i0-less lfinite-lmap llast-lmap llength-eq-0 llist-of-nellist-not-lnull lmap-nmap-inverse*)
have *131*: *llength* (*lconcat* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))) > *0*
using *126 127 128 129 130 i0-less llength-eq-0 lnull-lappend* **by** *auto*
have *13*: ¬ *lnull* (*lconcat* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))))
using *131 llength-LNil lnull-def* **by** *auto*
have *14*: ¬ *lnull* (*llast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))
by (*simp add*: *4 llast-lmap*)
have *13*: *nellist-of-llist* (*lappend* (*lconcat* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))))
(*llast* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))) =
nappend (*nellist-of-llist* (*lconcat* (*lbutlast* (*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))))
(*nellist-of-llist* (*llast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))

```
    using 13 14 by (metis nappend.abs-eq)
 have 14: (nellist-of-llist (llast (lmap llist-of-nellist (llist-of-nellist ll)))) =
         (nlast ll)
    by (simp add: 4 assms(4) llast-lmap nlast-llast)
 have 140:  llist-of-nellist (nconcat (nbutlast (nmap nbutlast ll))) =
           lconcat ((lmap llist-of-nellist ∘ llist-of-nellist) (nbutlast (nmap nbutlast ll)))
   using llist-of-nellist-nconcat[of (nbutlast (nmap (nbutlast) ll)) ]
    by blast
 have 141: ((lmap llist-of-nellist ∘ llist-of-nellist) (nbutlast (nmap nbutlast ll))) =
          lmap llist-of-nellist (llist-of-nellist (nbutlast (nmap nbutlast ll)))
    by simp
 have 142: (nbutlast (nmap nbutlast ll)) = nmap nbutlast (nbutlast ll)
    using   nmap-nbutlast[of nbutlast ll]      by simp
 have 143: (llist-of-nellist (nmap nbutlast (nbutlast ll))) = lmap nbutlast (llist-of-nellist (nbutlast ll))
      using lmap-llist-of-nellist[of nbutlast (nbutlast ll)] by simp
 have 144: (llist-of-nellist (nbutlast ll)) = lbutlast (llist-of-nellist ll)
       unfolding nbutlast-def apply simp
       using 126 by auto
 have 145: lmap nbutlast (llist-of-nellist (nbutlast ll)) = lmap nbutlast (lbutlast (llist-of-nellist ll))
    using 144 by simp
 have 146: lmap llist-of-nellist (lmap nbutlast (lbutlast (llist-of-nellist ll))) =
          lmap (llist-of-nellist ∘ nbutlast) (lbutlast (llist-of-nellist ll))
       by (simp add: llist.map-comp)
 have 147: lmap (llist-of-nellist ∘ nbutlast) (lbutlast (llist-of-nellist ll)) =
          lmap (lbutlast ∘ llist-of-nellist) (lbutlast (llist-of-nellist ll))
       using 43 by (metis lmap-lbutlast)
 have 148: lmap (lbutlast ∘ llist-of-nellist) (lbutlast (llist-of-nellist ll)) =
          (lbutlast (lmap lbutlast (lmap llist-of-nellist (llist-of-nellist ll))))
    using 41 by (metis lmap-lbutlast)
 have 149:  (lbutlast (lmap lbutlast (lmap llist-of-nellist (llist-of-nellist ll)))) =
           ((lmap llist-of-nellist ∘ llist-of-nellist) (nbutlast (nmap nbutlast ll)))
  using 141 142 143 144 146 147 148 by argo
 have 15: (nellist-of-llist (lconcat (lbutlast (lmap lbutlast (lmap llist-of-nellist (llist-of-nellist ll)))))) =
         (nconcat (nbutlast (nmap (nbutlast) ll)))
  using 140 149 by (metis quotient-help-2)
 show ?thesis
 by (simp add: 11 13 14 15 nfusecat-def2)
qed

lemma nfusecat-nconcat-nmap-nbutlast:
 assumes ⋀ xx. xx ∈ nset ll ⟹ nlength xx >0
        ⋀ xx. xx ∈ nset ll ⟹ nfinite xx
         nlastnfirst ll
        ¬ nfinite ll
 shows   nfusecat ll = nconcat (nmap (nbutlast) ll)
proof −
 let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
 have 1: nfusecat ll = (nellist-of-llist ∘ lfusecat ∘ ?n2l) ll
  unfolding nfusecat-def2 by simp
 have 01: (nellist-of-llist ∘ lfusecat ∘ ?n2l) ll =
```

nellist-of-llist (*lfusecat* (*lmap llist-of-nellist* (*llist-of-nellist ll*)))
by *simp*
have *2*: *nconcat* (*nmap* (*nbutlast*) *ll*) =
(*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap nbutlast ll*)
unfolding *nconcat-def2* **by** *simp*
have *3*: ⋀*xx*. *xx* ∈ *nset ll* ⟹(*llist-of-nellist* (*nbutlast xx*)) = *lbutlast* (*llist-of-nellist xx*)
unfolding *nbutlast-def* **apply** *simp*
by (*metis assms*(*1*) *lbutlast.ctr*(*1*) *less-numeral-extra*(*3*) *llength-LNil llength-lbutlast llength-llist-of-nellist*)
have *4*: ¬*lfinite* (*llist-of-nellist ll*)
using *assms*(*4*) *nfinite-def* **by** *auto*
have *7*: (⋀*xx*. *xx* ∈ *lset* (*lmap llist-of-nellist* (*llist-of-nellist ll*)) ⟹ *1* < *llength xx*)
using *assms* **apply** *auto*
by (*metis epred-1 llength-llist-of-nellist llist-of-nellist.simps*(*1*) *nless-le not-lnull-llength*)
have *8*: (⋀*xx*. *xx* ∈ *lset* (*lmap llist-of-nellist* (*llist-of-nellist ll*)) ⟹ *lfinite xx*)
using *assms* **apply** *auto* **using** *nfinite-def* **by** *blast*
have *9*: *llastlfirst* (*lmap llist-of-nellist* (*llist-of-nellist ll*))
using *assms nlastnfirst.rep-eq* **by** *auto*
have *10*: ¬ *lfinite* (*lmap llist-of-nellist* (*llist-of-nellist ll*))
using *assms*
using *4 lfinite-lmap* **by** *blast*

have *40*: (*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap nbutlast ll*) =
nellist-of-llist (*lconcat* (*lmap llist-of-nellist* (*lmap nbutlast* (*llist-of-nellist ll*))))
by *simp*
have *41*: ((*lmap* (*lbutlast*∘*llist-of-nellist*) (*llist-of-nellist ll*))) =
((*lmap lbutlast* (*lmap llist-of-nellist* (*llist-of-nellist ll*))))
by (*simp add*: *llist.map-comp*)
have *411*: ((*lmap llist-of-nellist* (*lmap nbutlast* (*llist-of-nellist ll*)))) =
((*lmap* (*llist-of-nellist*∘ *nbutlast*) (*llist-of-nellist ll*)))
by (*simp add*: *llist.map-comp*)
have *42*: ⋀*xx*. *xx*∈ *lset* (*llist-of-nellist ll*) ⟹ *0*< *nlength xx*
using *assms*(*1*) **by** *auto*
have *420*: (⋀*z*. *z* ∈ *lset* (*llist-of-nellist ll*) ⟹ (*llist-of-nellist* ∘ *nbutlast*) *z* = (*lbutlast* ∘ *llist-of-nellist*) *z*)
using *3* **by** *auto*
have *43*: ((*lmap* (*llist-of-nellist*∘ *nbutlast*) (*llist-of-nellist ll*))) =
((*lmap* (*lbutlast*∘ *llist-of-nellist*) (*llist-of-nellist ll*)))
using *3 4 assms lbutlast-not-lfinite*[*of* (*lmap llist-of-nellist* (*llist-of-nellist ll*))]
using *lset-llist-of-nellist-a*[*of ll*] **using** *comp-apply llist.map-cong0*[*of* (*llist-of-nellist ll*)
(*llist-of-nellist*∘ *nbutlast*) (*lbutlast*∘ *llist-of-nellist*)]
by *auto*
have *6*: (*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap nbutlast ll*) =
nellist-of-llist (*lconcat* (*lmap* (*lbutlast*∘*llist-of-nellist*) (*llist-of-nellist ll*)))
using *40 411 43* **by** *presburger*
have *11*: (*lfusecat* (*lmap llist-of-nellist* (*llist-of-nellist ll*))) =
(*lconcat* (*lmap* (*lbutlast*∘*llist-of-nellist*) (*llist-of-nellist ll*)))
using *lfusecat-lconcat-lmap-lbutlast*[*of* (*lmap llist-of-nellist* (*llist-of-nellist ll*))]
by (*metis 10 7 8 9 llist.map-comp*)
have *12*: *nellist-of-llist* (*lfusecat* (*lmap llist-of-nellist* (*llist-of-nellist ll*))) =
nellist-of-llist ((*lconcat* (*lmap* (*lbutlast*∘*llist-of-nellist*) (*llist-of-nellist ll*))))
using *11* **by** *simp*

show *?thesis*
by (*metis 01 11 6 nconcat-def2 nfusecat-def2*)
qed

lemma *nmap-nfilter-nmap-nbutlast-swap-finite*:
assumes *nfinite nells*
$\forall$ *nell* $\in$ *nset* (*nells*). *P* (*nfirst nell*)
$\bigwedge$ *xx*. *xx* $\in$ *nset nells* $\Longrightarrow$ *nlength xx* > *0*
$\bigwedge$ *xx*. *xx* $\in$ *nset nells* $\Longrightarrow$ *nfinite xx*
nlastnfirst nells
P (*nlast* (*nlast nells*))
shows (*nmap* (*nfilter P*) (*nmap* (*nbutlast*) *nells*)) =
(*nmap* (*nbutlast*) (*nmap* (*nfilter P*) *nells*))
using *assms*
proof (*induction nells rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
proof −
have *1*: *P* (*nnth y 0*)
using *NNil* **by** (*metis nellist.set-intros*(*1*) *nfirst-eq-nnth-zero*)
have *2*: *nmap* (*nfilter P*) (*nmap nbutlast* (*NNil y*)) = *NNil* (*nfilter P* (*nbutlast y*))
by *simp*
have *3*: *nlength y* > *0*
using *NNil.prems* **by** *simp*
have *4*: *nmap nbutlast* (*nmap* (*nfilter P*) (*NNil y*)) = *NNil* (*nbutlast* (*nfilter P y*))
by *simp*
have *5*: *P* (*nlast y*)
using *NNil.prems*(*5*) **by** *auto*
have *6*: $\exists$ *y1 z y2*. *y* = *nappend* (*NNil y1*) (*NNil y2*) $\vee$ *y* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
using *3*
by (*metis i0-less nappend-nbutlast-nlast-id nellist.collapse*(*1*) *nellist.exhaust-sel nfuse-NCons-b nfuse-leftneutral nlength-NNil*)
obtain *y1 z y2* **where** *9*: *y* = *nappend* (*NNil y1*) (*NNil y2*) $\vee$ *y* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
using *6* **by** *blast*
have *7*: *y* = *nappend* (*NNil y1*) (*NNil y2*) $\Longrightarrow$ (*nfilter P* (*nbutlast y*)) = (*nbutlast* (*nfilter P y*))
using *1 NNil.prems*(*5*) **apply** *simp*
by (*metis nappend-NNil nbutlast-snoc nfilter-NNil*)
have *8*: *y* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) $\Longrightarrow$(*nfilter P* (*nbutlast y*)) = (*nbutlast* (*nfilter P y*))
proof −
assume *a0*: *y* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
show (*nfilter P* (*nbutlast y*)) = (*nbutlast* (*nfilter P y*))
proof −
have *111*: *nbutlast y* = (*NCons y1 z*)
using *a0 5 6 NNil.prems nbutlast-eq-NCons-conv*[*of y y1 z*]
by (*metis nappend-NCons nappend-NNil nbutlast-snoc*)
have *112*: $\exists$ *x*$\in$*nset z*. *P x* $\Longrightarrow$ *nfilter P* (*NCons y1 z*) = (*if P y1 then* (*NCons y1* (*nfilter P z*)) *else nfilter P z*)
using *assms nfilter-NCons*[*of z P y1*] **by** *blast*

have *113*: *P y1*
using *NNil.prems 1 a0* **by** *auto*
have *114*: *P y2*
using *NNil.prems*
using *a0* **by** (*metis nfinite-NNil nfinite-nappend nfusecat-NNil nfusecat-nfinite nlast-NNil nlast-nappend*)
have *115*: ∃ *x*∈*nset* (*NCons y1 z*). *P x*
by (*simp add*: *113*)
have *116*: *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) = (*nappend* (*nappend* (*NNil y1*) *z*) (*NNil y2*))
by *simp*
have *117*: (*nfilter P y*) = *nappend* (*nfilter P* ((*nappend* (*NNil y1*) *z*))) (*nfilter P* (*NNil y2*))
using *113 114 116 5 a0 nappend-nfilter-nfinite*[*of* ((*nappend* (*NNil y1*) *z*)) *P* (*NNil y2*)]
using *NNil.prems*
by (*metis nfinite-nappend split-nellist-a*)
show *?thesis*
by (*simp add*: *111 117*)
qed
qed
show *?thesis*
using *7 8 9* **by** *auto*
qed
next
case (*NCons y nell*)
then show *?case*
proof −
have *1*: *P* (*nnth y 0*)
using *NCons.prems* **by** (*metis nellist.set-intros*(*2*) *nfirst-eq-nnth-zero*)
have *2*: *nmap* (*nfilter P*) (*nmap nbutlast* (*NCons y nell*)) =
NCons (*nfilter P* (*nbutlast y*)) (*nmap* (*nfilter P*) (*nmap nbutlast nell*))
by *simp*
have *3*: *nmap nbutlast* (*nmap* (*nfilter P*) (*NCons y nell*)) =
NCons (*nbutlast* (*nfilter P y*)) (*nmap nbutlast* (*nmap* (*nfilter P*) *nell*))
by *simp*
have *4*: *nlength y* > *0*
using *NCons.prems* **by** *simp*
have *5*: *P* (*nlast y*)
using *NCons.prems*
using *NCons.hyps* **by** (*metis insert-iff nellist.set*(*2*) *nfinite-nrev nlast-nrev nlastnfirst-LCons nset-nlast nset-nrev*)
have *6*: ∃ *y1 z y2*. *y* = *nappend* (*NNil y1*) (*NNil y2*) ∨ *y* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
using *4* **by** (*metis nappend-NNil nappend-nbutlast-nlast-id nellist.collapse*(*1*) *nellist-split-2-first*)
obtain *y1 z y2* **where** *9*: *y* = *nappend* (*NNil y1*) (*NNil y2*) ∨ *y* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
using *6* **by** *blast*
have *7*: *y* = *nappend* (*NNil y1*) (*NNil y2*) ⟹ (*nfilter P* (*nbutlast y*)) = (*nbutlast* (*nfilter P y*))
using *1 NCons.prems* **apply** *simp*
using *5* **by** (*metis nappend-NNil nbutlast-snoc nellist.set-intros*(*1*) *nfilter-NCons nfilter-NNil nlast-NCons nlast-NNil*)
have *8*: *y* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) ⟹(*nfilter P* (*nbutlast y*)) = (*nbutlast* (*nfilter P y*))

```
      proof −
       assume a0: y = nappend (NNil y1) (nappend z (NNil y2))
       show (nfilter P (nbutlast y)) = (nbutlast (nfilter P y))
       proof −
         have 111: nbutlast y = (NCons y1 z)
         using a0 5 6 NCons.prems nbutlast-eq-NCons-conv[of y y1 z ]
         by (metis nappend-NCons nappend-NNil nbutlast-snoc)
        have 112: ∃ x∈nset z. P x ⟹ nfilter P (NCons y1 z) = (if P y1 then (NCons y1 (nfilter P z)) else
nfilter P z)
         using assms nfilter-NCons[of z P y1]  by blast
        have 113:  P y1
         using NCons.prems  1 a0 by auto
        have 114:  P y2
            using NCons.prems
          using 111 5 a0 by (metis nbutlast-nfinite nellist.set-intros(2) nfinite-NCons nfinite-NNil nlast-NNil
nlast-nappend)
        have 115: ∃ x∈nset (NCons y1 z). P x
          by (simp add: 113)
        have 116: nappend (NNil y1) (nappend z (NNil y2)) = (nappend (nappend (NNil y1) z) (NNil y2))
            by simp
        have 117: (nfilter P y) = nappend (nfilter P ((nappend (NNil y1) z))) (nfilter P (NNil y2))
         using 113 114 116 5 a0 nappend-nfilter-nfinite[of ((nappend (NNil y1) z)) P (NNil y2)  ]
           using NCons.prems
              by (metis nellist.set-intros(2) nfinite-nappend split-nellist-a)
        show ?thesis
        by (simp add: 111 117)
      qed
    qed
    have 12: nfilter P (nbutlast y) = nbutlast (nfilter P y)
      using 7 8 9 by blast
    have 16: y = nappend (NNil y1) (NNil y2) ⟹    ?thesis
       using 12 2 3 NCons.IH NCons.prems(1,2,3,4,5) by (metis nellist.set-intros(3) nlast-NCons nlastn-
first-LCons)
    have 20: y = nappend (NNil y1) (nappend z (NNil y2)) ⟹ ?thesis
       using 8 NCons.IH NCons.prems(1,2,3,4,5) by auto
    show ?thesis
    using 16 20 9 by blast
   qed
qed

lemma nmap-nfilter-nmap-nbutlast-swap:
 assumes
        ∀ nell ∈ nset (nells). P (nfirst nell)
        ⋀ xx. xx ∈ nset nells ⟹ nlength xx >0
        ⋀ xx. xx ∈ nset nells ⟹ nfinite xx
         nlastnfirst nells
        ¬ nfinite nells
 shows  (nmap (nfilter P) (nmap (nbutlast) nells)) =
        (nmap (nbutlast) (nmap (nfilter P) nells))
using assms
```

proof (*coinduction arbitrary*: *nells*)
case (*Eq-nellist nells1*)
then show *?case*
proof −
have *1*: *is-NNil* (*nmap* (*nfilter P*) (*nmap nbutlast nells1*)) = *is-NNil* (*nmap nbutlast* (*nmap* (*nfilter P*) *nells1*))
by *simp*
have *2*: (*is-NNil* (*nmap* (*nfilter P*) (*nmap nbutlast nells1*)) ⟶
is-NNil (*nmap nbutlast* (*nmap* (*nfilter P*) *nells1*)) ⟶ *nlast* (*nmap* (*nfilter P*) (*nmap nbutlast nells1*)) = *nlast* (*nmap nbutlast* (*nmap* (*nfilter P*) *nells1*)))
using *Eq-nellist*(*5*) **by** *force*
have *3*: (¬ *is-NNil* (*nmap* (*nfilter P*) (*nmap nbutlast nells1*)) ⟶
¬ *is-NNil* (*nmap nbutlast* (*nmap* (*nfilter P*) *nells1*)))
by *simp*
have *4*: ∃ *x xs*. *nells1* = (*NCons x xs*)
by (*metis Eq-nellist*(*5*) *is-NNil-nappend nappend-inf nellist.collapse*(*2*))
obtain *x xs* **where** *5*: *nells1* = (*NCons x xs*)
using *4* **by** *blast*
have *6*: *P* (*nlast x*)
by (*metis 5 Eq-nellist*(*1*,*4*,*5*) *enat-ord-simps*(*3*) *in-nset-conv-nnth nfinite-conv-nlength-enat nlast-NNil nlastnfirst-LCons nnth-Suc-NCons not-infinity-eq ntaken-0*
ntaken-nlast)
have *7*: *nlength x* > *0*
using *5 Eq-nellist*(*2*) **by** *auto*
have *8*: ∃ *y1 z y2*. *x* = *nappend* (*NNil y1*) (*NNil y2*) ∨ *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
by (*metis 7 nappend-NNil nappend-nbutlast-nlast-id nellist.collapse*(*1*) *nellist-split-2-first*)
obtain *y1 z y2* **where** *9*: *x* = *nappend* (*NNil y1*) (*NNil y2*) ∨ *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
using *8* **by** *blast*
have *10*: *x* = *nappend* (*NNil y1*) (*NNil y2*) ⟹ *nfilter P* (*nbutlast x*) = *nbutlast* (*nfilter P x*)
using *5 6 Eq-nellist* **apply** *simp*
by (*metis nappend-NNil nbutlast-snoc nfilter-NNil*
nfirst-eq-nnth-zero nnth-0)
have *11*: *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) ⟹ *nfilter P* (*nbutlast x*) = *nbutlast* (*nfilter P x*)
proof −
assume *a0*: *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*))
show *nfilter P* (*nbutlast x*) = *nbutlast* (*nfilter P x*)
proof −
have *111*: *nbutlast x* = (*NCons y1 z*)
using *a0 5 6 Eq-nellist nbutlast-eq-NCons-conv*[*of x y1 z*]
by (*metis nappend-NCons nappend-NNil nbutlast-snoc*)
have *112*: ∃*x*∈*nset z*. *P x* ⟹ *nfilter P* (*NCons y1 z*) = (*if P y1 then* (*NCons y1* (*nfilter P z*)) *else nfilter P z*)
using *assms nfilter-NCons*[*of z P y1*] **by** *blast*
have *113*: *P y1*
by (*metis 5 Eq-nellist*(*1*) *a0 nappend-NNil nellist.set-intros*(*2*) *nfirst-eq-nnth-zero nnth-0*)
have *114*: *P y2*
by (*metis 5 6 Eq-nellist*(*3*) *a0 nellist.set-intros*(*2*) *nfinite-nappend nlast-NNil nlast-nappend*)
have *115*: ∃*x*∈*nset* (*NCons y1 z*). *P x*
by (*simp add*: *113*)

have *116*: *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) = (*nappend* (*nappend* (*NNil y1*) *z*) (*NNil y2*))
by *simp*
have *117*: (*nfilter P x*) = *nappend* (*nfilter P* ((*nappend* (*NNil y1*) *z*))) (*nfilter P* (*NNil y2*))
using *113 114 116 5 a0 nappend-nfilter-nfinite*[*of* ((*nappend* (*NNil y1*) *z*)) *P* (*NNil y2*)]
using *Eq-nellist*(*3*)
by (*metis nellist.set-intros*(*2*) *nfinite-nappend split-nellist-a*)
show *?thesis*
by (*simp add*: *111 117*)
qed
qed
have *12*: *nfilter P* (*nbutlast x*) = *nbutlast* (*nfilter P x*)
using *10 11 9* **by** *blast*
have *13*: *x* = *nappend* (*NNil y1*) (*NNil y2*) ⟹
ntl (*nmap* (*nfilter P*) (*nmap nbutlast nells1*)) = *nmap* (*nfilter P*) (*nmap nbutlast xs*)
using *5 6 Eq-nellist* **by** *simp*
have *14*: *x* = *nappend* (*NNil y1*) (*NNil y2*) ⟹
ntl (*nmap nbutlast* (*nmap* (*nfilter P*) *nells1*)) = *nmap nbutlast* (*nmap* (*nfilter P*) *xs*)
using *5 6 Eq-nellist* **by** *simp*
have *15*: *x* = *nappend* (*NNil y1*) (*NNil y2*) ⟹
(∀ *nell*∈*nset xs*. *P* (*nfirst nell*)) ∧ (∀ *xx*. *xx* ∈ *nset xs* ⟶ *0* < *nlength xx*) ∧ (∀ *xx*. *xx* ∈ *nset xs* ⟶ *nfinite xx*) ∧ *nlastnfirst xs* ∧ ¬ *nfinite xs*
using *5 6 Eq-nellist* **by** *simp*
have *16*: *x* = *nappend* (*NNil y1*) (*NNil y2*) ⟹ *?thesis*
using *10 13 15 5* **by** *auto*
have *17*: *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) ⟹
ntl (*nmap* (*nfilter P*) (*nmap nbutlast nells1*)) = *nmap* (*nfilter P*) (*nmap nbutlast xs*)
using *5 6 Eq-nellist* **by** *simp*
have *18*: *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) ⟹
ntl (*nmap nbutlast* (*nmap* (*nfilter P*) *nells1*)) = *nmap nbutlast* (*nmap* (*nfilter P*) *xs*)
using *5 6 Eq-nellist* **by** *simp*
have *19*: *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) ⟹
(∀ *nell*∈*nset xs*. *P* (*nfirst nell*)) ∧ (∀ *xx*. *xx* ∈ *nset xs* ⟶ *0* < *nlength xx*) ∧ (∀ *xx*. *xx* ∈ *nset xs* ⟶ *nfinite xx*) ∧ *nlastnfirst xs* ∧ ¬ *nfinite xs*
using *5 6 Eq-nellist* **by** *simp*
have *20*: *x* = *nappend* (*NNil y1*) (*nappend z* (*NNil y2*)) ⟹ *?thesis*
using *11 19 5* **by** *auto*
show *?thesis*
using *16 20 9* **by** *fastforce*
qed
qed

lemma *nfilter-nconcat-nfinite-alt*:
assumes ∀ *nell*∈*nset nells*. *nfinite nell*
∀ *nell* ∈ *nset nells*. *P* (*nfirst nell*)
shows *nfilter P* (*nconcat nells*) = *nconcat* (*nmap* (*nfilter P*) *nells*)
proof −
have *1*: ∀ *nell* ∈ *nset nells*. (∃ *x* ∈ *nset nell*. *P x*)
using *assms* **apply** *auto*
by (*metis nfinite-nrev nlast-nrev nset-nlast nset-nrev*)
show *?thesis* **using** *1 nfilter-nconcat-nfinite*[*of nells*]

using *assms* **by** *blast*
qed

lemma *in-nset-nbutlast-finite*:
assumes $\bigwedge$ *nell*. *nell* ∈ *nset* (*nells*)⟹ *P* (*nfirst nell*)
$\bigwedge$ *xx*. *xx* ∈ *nset nells* ⟹ *nlength xx* >*0*
$\bigwedge$ *xx*. *xx* ∈ *nset nells* ⟹ *nfinite xx*
nlastnfirst nells
nfinite nells
P (*nlast* (*nlast nells*))
nell ∈ *nset* (*nmap nbutlast nells*)
shows *P* (*nfirst nell*)
proof −
have *1*: ∃ *yy*. *nell* = *nbutlast yy* ∧ *yy* ∈ *nset nells*
using *assms*
using *assms*(*7*) **by** (*metis in-nset-conv-nnth nlength-nmap nnth-nmap*)
obtain *yy* **where** *2*: *nell* = *nbutlast yy* ∧ *yy* ∈ *nset nells*
using *1* **by** *blast*
have *3*: *nfirst* (*yy*) = *nfirst nell*
by (*metis 2 assms*(*2*,*3*) *is-NNil-nappend nappend-NNil*
nellist-split-2-first nfirst-eq-nnth-zero nnth-nappend1
snocs-eq-iff-nbutlast zero-enat-def zero-le)
show *?thesis*
using *2 3 assms*(*1*) **by** *fastforce*
qed

lemma *in-nset-nbutlast*:
assumes $\bigwedge$ *nell*. *nell* ∈ *nset* (*nells*)⟹ *P* (*nfirst nell*)
$\bigwedge$ *xx*. *xx* ∈ *nset nells* ⟹ *nlength xx* >*0*
$\bigwedge$ *xx*. *xx* ∈ *nset nells* ⟹ *nfinite xx*
nlastnfirst nells
¬ *nfinite nells*
nell ∈ *nset* (*nmap nbutlast nells*)
shows *P* (*nfirst nell*)
proof −
have *1*: ∃ *yy*. *nell* = *nbutlast yy* ∧ *yy* ∈ *nset nells*
using *assms*(*6*) **by** (*metis in-nset-conv-nnth nlength-nmap nnth-nmap*)
obtain *yy* **where** *2*: *nell* = *nbutlast yy* ∧ *yy* ∈ *nset nells*
using *1* **by** *blast*
have *3*: *nfirst* (*yy*) = *nfirst nell*
by (*metis 2 assms*(*2*,*3*) *is-NNil-nappend nappend-NNil*
nellist-split-2-first nfirst-eq-nnth-zero nnth-nappend1
snocs-eq-iff-nbutlast zero-enat-def zero-le)
show *?thesis*
using *2 3 assms*(*1*) **by** *fastforce*
qed

lemma *nfilter-nfusecat-dist-zero*:
assumes *nfinite nells*
$\bigwedge$ *nell*. *nell* ∈ *nset* (*nells*)⟹ *P* (*nfirst nell*)

$\bigwedge xx.\ xx \in nset\ nells \Longrightarrow nlength\ xx > 0$
$\bigwedge xx.\ xx \in nset\ nells \Longrightarrow nfinite\ xx$
nlastnfirst nells
P (nlast (nlast nells))
nlength nells = 0
shows *nfilter P (nfusecat nells) = nfusecat (nmap (nfilter P) nells)*
using *assms*
by (*metis NNil-eq-ntake-iff nellist.simps(14) nfusecat-NNil ntake-all zero-le*)

lemma *nfilter-nfusecat-dist-finite*:
assumes *nfinite nells*
$\bigwedge nell.\ nell \in nset\ (nells) \Longrightarrow P\ (nfirst\ nell)$
$\bigwedge xx.\ xx \in nset\ nells \Longrightarrow nlength\ xx > 0$
$\bigwedge xx.\ xx \in nset\ nells \Longrightarrow nfinite\ xx$
nlastnfirst nells
P (nlast (nlast nells))
nlength nells > 0
shows *nfilter P (nfusecat nells) = nfusecat (nmap (nfilter P) nells)*
proof −
have *1*: *nfilter P (nfusecat nells) =*
nfilter P (nappend (nconcat (nbutlast (nmap nbutlast nells))) (nlast nells))
using *assms nfusecat-nconcat-nmap-nbutlast-finite*[*of nells*] **by** *presburger*
have *2*: $\forall nell \in nset\ (nmap\ nbutlast\ nells).\ nfinite\ nell$
by (*simp add*: *assms nbutlast-nfinite nellist.set-map*)
have *20*: $\forall nell \in nset\ (nmap\ nbutlast\ nells).\ P\ (nfirst\ nell)$
using *assms in-nset-nbutlast-finite* **by** *blast*
have *21*: $\exists x \in nset\ (nappend\ (nconcat\ (nbutlast\ (nmap\ nbutlast\ nells)))\ (nlast\ nells)).\ P\ x$
using *assms* **by** (*metis nfusecat-nconcat-nmap-nbutlast-finite nfusecat-nfinite nlastfirst-nfusecat-nlast nset-nlast*)
have *22*: $\exists x \in nset\ (nconcat\ (nbutlast\ (nmap\ nbutlast\ nells))).\ P\ x$
using *assms* **apply** *simp*
using *20*
by (*metis in-nset-conv-nnth in-nset-nbutlastD nfilter-nconcat-nfinite-help nfirst-eq-nnth-zero zero-enat-def zero-le*)
have *23*: $\exists x \in nset\ (nlast\ nells).\ P\ x$
using *assms* **by** (*meson nset-nlast*)
have *24*: *nfinite (nconcat (nbutlast (nmap nbutlast nells)))*
using *2 assms* **by** (*metis in-nset-nbutlastD nbutlast-nfinite nfinite-nconcat nfinite-nmap*)
have *25*: *nfilter P (nappend (nconcat (nbutlast (nmap nbutlast nells))) (nlast nells)) =*
nappend (nfilter P (nconcat (nbutlast (nmap nbutlast nells)))) (nfilter P (nlast nells))
using *nfilter-nappend*[*of (nconcat (nbutlast (nmap nbutlast nells))) (nlast nells) P*]
using *21 22 23 24* **by** *presburger*
have *26*: *all-nfinite (nmap nbutlast nells)*
by (*simp add*: *2*)
have *27*: $\forall nell \in nset\ (nmap\ nbutlast\ nells).\ P\ (nfirst\ nell)$
by (*simp add*: *20*)
have *28*: *all-nfinite (nmap nbutlast nells)* $\Longrightarrow$
$\forall nell \in nset\ (nmap\ nbutlast\ nells).\ P\ (nfirst\ nell) \Longrightarrow$
nfilter P (nconcat (nmap nbutlast nells)) = nconcat (nmap (nfilter P) (nmap nbutlast nells))
using *nfilter-nconcat-nfinite-alt* **by** *auto*

let *?ll1=nfilter P (nconcat (nmap nbutlast nells))*
let *?ll2=(nconcat (nmap (nfilter P) (nmap nbutlast nells)))*
let *?ll3=(nfilter P (nlast nells))*
have *29*: *?ll1 = ?ll2*
using *26 27 28* **by** *blast*
have *31*: *?ll1 = ?ll2 ⟹ (nappend ?ll1 ?ll3 = nappend ?ll2 ?ll3)*
by *simp*
have *3*: *nappend ?ll1 ?ll3 = nappend ?ll2 ?ll3*
using *29 31* **by** *auto*
have *4*: *(nmap (nfilter P) (nmap (nbutlast) nells)) =*
(nmap (nbutlast) (nmap (nfilter P) nells))
using *assms nmap-nfilter-nmap-nbutlast-swap-finite* **by** *blast*
have *5*: *nconcat (nmap (nfilter P) (nmap (nbutlast) nells)) =*
nconcat (nmap (nbutlast) (nmap (nfilter P) nells))
using *4* **by** *simp*
have *50*: *⋀ i. nfinite (nnth (nmap (nfilter P) nells) i)*
using *assms in-nset-conv-nnth*[*of - (nmap (nfilter P) nells)*]
by (*metis in-nset-conv-nnth length-nfilter-le linorder-linear nfinite-nlength-enat nfinite-ntaken nlast-nmap nlength-nmap nnth-nmap ntaken-all ntaken-nlast*)
have *6*: *(⋀xx. xx ∈ nset (nmap (nfilter P) nells) ⟹ nfinite xx)*
using *assms 50 all-nfinite-nnth-a* **by** *blast*
have *7*: *⋀ nell. nell ∈ nset (nells)⟹ P (nlast nell)*
using *assms*
proof (*induction nells rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *auto*
next
case (*NCons x nell*)
then show *?case* **apply** *auto*
apply (*metis nfinite-nrev nlast-nrev nset-nlast nset-nrev*)
by (*metis ile0-eq in-nset-conv-nnth nnth-nlast the-enat.simps*)
qed

have *8*: *(⋀xx. xx ∈ nset (nmap (nfilter P) nells) ⟹ 0 < nlength xx)*
proof −
fix *xx*
assume *a0*: *xx ∈ nset (nmap (nfilter P) nells)*
show *0 < nlength xx*
proof −
have *80*: *∃ yy. xx= nfilter P yy ∧ yy ∈ nset nells*
using *assms*
by (*metis a0 image-iff nellist.set-map*)
obtain *yy* **where** *81*: *xx= nfilter P yy ∧ yy ∈ nset nells*
using *80* **by** *blast*
have *82*: *P (nfirst yy)*
by (*simp add*: *81 assms*)
have *83*: *nlength yy > 0*
using *assms* **using** *81* **by** *blast*
have *84*: *P (nlast yy)*
by (*simp add*: *7 81*)

have *85*: $\exists z.\ yy = nappend\ (NNil\ (nfirst\ yy))\ (NNil\ (nlast\ yy)) \vee yy = nappend\ (NNil\ (nfirst\ yy))\ (nappend\ z\ (NNil\ (nlast\ yy)))$
using *assms*
by (*metis 81 i0-less nappend-nbutlast-nlast-id nfuse-def1 nfuse-leftneutral nlength-NNil ntl-nappend*)
obtain *z* **where** *86*: $yy = nappend\ (NNil\ (nfirst\ yy))\ (NNil\ (nlast\ yy)) \vee yy = nappend\ (NNil\ (nfirst\ yy))\ (nappend\ z\ (NNil\ (nlast\ yy)))$
using *85* **by** *blast*
have *87*: $yy = nappend\ (NNil\ (nfirst\ yy))\ (NNil\ (nlast\ yy)) \Longrightarrow nlength\ (nfilter\ P\ yy) > 0$
apply *simp*
by (*metis 82 83 84 i0-less nellist.set-intros*(*1*)
nfilter-NCons nfilter-NNil)
have *88*: $yy = nappend\ (NNil\ (nfirst\ yy))\ (nappend\ z\ (NNil\ (nlast\ yy))) \Longrightarrow nlength\ (nfilter\ P\ yy) > 0$
proof −
assume *a0*: $yy = nappend\ (NNil\ (nfirst\ yy))\ (nappend\ z\ (NNil\ (nlast\ yy)))$
show $nlength\ (nfilter\ P\ yy) > 0$
proof −
have *880*: $(nfilter\ P\ yy) = nappend\ (nfilter\ P\ (NNil\ (nfirst\ yy)))\ (nfilter\ P\ (nappend\ z\ (NNil\ (nlast\ yy))))$
using *81 82 84 a0 assms nappend-nfilter-nfinite nfinite-nappend split-nellist-a* **by** *metis*
have *881*: $(nfilter\ P\ (NNil\ (nfirst\ yy))) = (NNil\ (nfirst\ yy))$
by *simp*
have *882*: $nlength\ (nappend\ (nfilter\ P\ (NNil\ (nfirst\ yy)))\ (nfilter\ P\ (nappend\ z\ (NNil\ (nlast\ yy))))) > 0$
by *simp*
show *?thesis*
by (*simp add*: *880*)
qed
qed
have *89*: $nlength\ (nfilter\ P\ yy) > 0$
using *87 88 86* **by** *auto*
show *?thesis* **using** *89 81* **by** *simp*
qed
qed
have *9*: $nlastnfirst\ (nmap\ (nfilter\ P)\ nells)$
unfolding *nlastnfirst-def1*
proof *auto*
fix *i*
assume *a1*: $enat\ (Suc\ i) \leq nlength\ nells$
show $nlast\ (nnth\ (nmap\ (nfilter\ P)\ nells)\ i) = nfirst\ (nfilter\ P\ (nnth\ nells\ (Suc\ i)))$
proof −
have *90*: $P\ (nfirst\ (nnth\ nells\ (Suc\ i)))$
by (*meson a1 assms in-nset-conv-nnth*)
have *91*: $nfirst\ (nfilter\ P\ (nnth\ nells\ (Suc\ i))) = (nfirst\ (nnth\ nells\ (Suc\ i)))$
by (*metis 90 ndropn-nfuse nfilter-nfirst nfinite-NNil nlast-NNil*)
have *92*: $nlast\ (nnth\ (nmap\ (nfilter\ P)\ nells)\ i) = nlast\ (nfilter\ P\ (nnth\ nells\ i))$
using *Suc-ile-eq a1* **by** *auto*
have *93*: $P\ (nlast\ (nnth\ nells\ i))$
using *90 a1 assms nlastnfirst-def1* **by** *fastforce*
have *930*: $nfinite\ (nnth\ nells\ i)$
using *a1 assms* **by** (*metis Suc-ile-eq all-nfinite-nnth-b nless-le*)

```
    have 931: (∃ n. nlength (nnth nells i) = enat n)
      using 930 nfinite-conv-nlength-enat by auto
    have 932: (nnth nells i) = (ntaken ((the-enat(nlength (nnth nells i)))) (nnth nells i) )
      using 931 by (metis min.cobounded1 min.idem ntaken-all the-enat.simps)
    have 933: nlast (nfilter P (ntaken (the-enat (nlength (nnth nells i))) (nnth nells i))) =
          nlast (ntaken (the-enat (nlength (nnth nells i))) (nnth nells i))
     using nfilter-nlast[of ((the-enat(nlength (nnth nells i)))) nnth nells i P ]
      using 93 931 932 by force
    have 94: nlast (nfilter P (nnth nells i)) = (nlast (nnth nells i))
     using 932 933 by auto
    have 95: (nlast (nnth nells i)) = (nfirst (nnth nells (Suc i)))
     using a1 assms nlastnfirst-def1  by blast
    show ?thesis
    by (simp add: 91 92 94 95)
   qed
  qed
 have 10:  nfinite (nmap (nfilter P) nells)
   using assms
   using nfinite-nmap by blast
 have 9: nappend (nconcat (nbutlast (nmap nbutlast (nmap (nfilter P) nells)))) (nlast (nmap (nfilter P)
nells)) =
        nfusecat (nmap (nfilter P) nells)
  using assms  nfusecat-nconcat-nmap-nbutlast-finite[of (nmap (nfilter P) nells)]
  using 6 8 9 by force
 have 11: (nfilter P (nlast nells)) = (nlast (nmap (nfilter P) nells))
   by (simp add: assms(1))

 show ?thesis
  using 1 3 4 9 25 11
  using 24 27 by (metis (no-types, lifting) in-nset-nbutlastD nfilter-nconcat-nfinite-alt nfinite-nconcat
nmap-nbutlast)
qed

lemma nfilter-nfusecat-dist:
 assumes
       ⋀ nell. nell ∈ nset (nells)⟹ P (nfirst nell)
       ⋀ xx. xx ∈ nset nells ⟹ nlength xx >0
       ⋀ xx. xx ∈ nset nells ⟹ nfinite xx
        nlastnfirst nells
       ¬ nfinite nells
 shows nfilter P (nfusecat nells) = nfusecat (nmap (nfilter P) nells)
proof −
 have 1: nfilter P (nfusecat nells) = nfilter P (nconcat (nmap (nbutlast) nells))
    by (metis assms(2,3,4,5)       nfusecat-nconcat-nmap-nbutlast)
 have 2: ∀ nell∈nset (nmap nbutlast nells). nfinite nell
  by (simp add: assms(3) nbutlast-nfinite nellist.set-map)
 have 20: ∀ nell∈nset (nmap nbutlast nells). P (nfirst nell)
  using assms(1,2,3,4,5) in-nset-nbutlast by blast

 have 3: nfilter P (nconcat (nmap (nbutlast) nells)) =
```

nconcat (nmap (nfilter P) (nmap (nbutlast) nells))
using *nfilter-nconcat-nfinite-alt*[*of* (*nmap nbutlast nells*) *P*]
using *2 assms 20* **by** *blast*
have *4*: (*nmap* (*nfilter P*) (*nmap* (*nbutlast*) *nells*)) =
(*nmap* (*nbutlast*) (*nmap* (*nfilter P*) *nells*))
using *assms*(*1,2,3,4,5*) *nmap-nfilter-nmap-nbutlast-swap* **by** *blast*
have *5*: *nconcat* (*nmap* (*nfilter P*) (*nmap* (*nbutlast*) *nells*)) =
nconcat (*nmap* (*nbutlast*) (*nmap* (*nfilter P*) *nells*))
using *4* **by** *simp*
have *50*: ⋀ *i*. *nfinite* (*nnth* (*nmap* (*nfilter P*) *nells*) *i*)
using *assms in-nset-conv-nnth*[*of* - (*nmap* (*nfilter P*) *nells*)]
by (*metis Suc-ile-eq enat-ord-simps*(*3*) *enat-the-enat in-nset-conv-nnth length-nfilter-le linorder-not-less nfinite-conv-nlength-enat nnth-nmap*)
have *6*: (⋀*xx*. *xx* ∈ *nset* (*nmap* (*nfilter P*) *nells*) ⟹ *nfinite xx*)
using *assms 50 all-nfinite-nnth-a* **by** *blast*
have *7*: ⋀ *nell*. *nell* ∈ *nset* (*nells*)⟹ *P* (*nlast nell*)
using *assms*
by (*metis enat-ord-simps*(*3*) *in-nset-conv-nnth nfinite-conv-nlength-enat nlastnfirst-def1 not-infinity-eq*)
have *8*: (⋀*xx*. *xx* ∈ *nset* (*nmap* (*nfilter P*) *nells*) ⟹ *0* < *nlength xx*)
proof −
fix *xx*
assume *a0*: *xx* ∈ *nset* (*nmap* (*nfilter P*) *nells*)
show *0* < *nlength xx*
proof −
have *80*: ∃ *yy*. *xx*= *nfilter P yy* ∧ *yy* ∈ *nset nells*
using *assms*
by (*metis a0 image-iff nellist.set-map*)
obtain *yy* **where** *81*: *xx*= *nfilter P yy* ∧ *yy* ∈ *nset nells*
using *80* **by** *blast*
have *82*: *P* (*nfirst yy*)
by (*simp add*: *81 assms*(*1*))
have *83*: *nlength yy* > *0*
using *assms* **using** *81* **by** *blast*
have *84*: *P* (*nlast yy*)
by (*simp add*: *7 81*)
have *85*: ∃ *z*. *yy* = *nappend* (*NNil* (*nfirst yy*)) (*NNil* (*nlast yy*)) ∨ *yy* = *nappend* (*NNil* (*nfirst yy*)) (*nappend z* (*NNil* (*nlast yy*)))
by (*metis 81 assms*(*2*) *i0-less nappend-nbutlast-nlast-id nfuse-def1 nfuse-leftneutral nlength-NNil ntl-nappend*)
obtain *z* **where** *86*: *yy* = *nappend* (*NNil* (*nfirst yy*)) (*NNil* (*nlast yy*)) ∨ *yy* = *nappend* (*NNil* (*nfirst yy*)) (*nappend z* (*NNil* (*nlast yy*)))
using *85* **by** *blast*
have *87*: *yy* = *nappend* (*NNil* (*nfirst yy*)) (*NNil* (*nlast yy*)) ⟹ *nlength* (*nfilter P yy*) > *0*
apply *simp*
by (*metis 82 83 84 i0-less nellist.set-intros*(*1*)
nfilter-NCons nfilter-NNil)
have *88*: *yy* = *nappend* (*NNil* (*nfirst yy*)) (*nappend z* (*NNil* (*nlast yy*))) ⟹ *nlength* (*nfilter P yy*) > *0*
proof −
assume *a0*: *yy* = *nappend* (*NNil* (*nfirst yy*)) (*nappend z* (*NNil* (*nlast yy*)))
show *nlength* (*nfilter P yy*) > *0*

```
        proof −
        have 880: (nfilter P yy) = nappend (nfilter P (NNil (nfirst yy))) (nfilter P (nappend z (NNil (nlast
yy))))
            using 81 82 84 a0 assms(3) nappend-nfilter-nfinite nfinite-nappend split-nellist-a  by metis
         have 881: (nfilter P (NNil (nfirst yy))) = (NNil (nfirst yy))
           by simp
        have 882: nlength (nappend (nfilter P (NNil (nfirst yy))) (nfilter P (nappend z (NNil (nlast yy)))))
>0
           by simp
         show ?thesis
         by (simp add: 880)
        qed
       qed
    have 89: nlength (nfilter P yy) > 0
      using 87 88 86 by auto
    show ?thesis using 89 81      by simp
   qed
  qed
 have 9: nlastnfirst (nmap (nfilter P) nells)
    unfolding nlastnfirst-def1
  proof auto
   fix i
   assume a1:  enat (Suc i) ≤ nlength nells
   show  nlast (nnth (nmap (nfilter P) nells) i) = nfirst (nfilter P (nnth nells (Suc i)))
   proof −
     have 90: P (nfirst (nnth nells (Suc i)))
      by (meson a1 assms(1) in-nset-conv-nnth)
     have 91: nfirst (nfilter P (nnth nells (Suc i))) = (nfirst (nnth nells (Suc i)))
       by (metis 90 ndropn-nfuse nfilter-nfirst nfinite-NNil nlast-NNil)
     have 92: nlast (nnth (nmap (nfilter P) nells) i) = nlast (nfilter P (nnth nells i))
        using Suc-ile-eq a1 by auto
     have 93: P (nlast (nnth nells i))
       using 90 a1 assms(4) nlastnfirst-def1 by fastforce
     have 930: nfinite (nnth nells i)
       using a1 assms(3) by (metis Suc-ile-eq all-nfinite-nnth-b nless-le)
     have 931: (∃ n. nlength (nnth nells i) = enat n)
        using 930 nfinite-conv-nlength-enat by auto
     have 932: (nnth nells i) = (ntaken ((the-enat(nlength (nnth nells i)))) (nnth nells i) )
        using 931 by (metis min.cobounded1 min.idem ntaken-all the-enat.simps)
     have 933: nlast (nfilter P (ntaken (the-enat (nlength (nnth nells i))) (nnth nells i))) =
                nlast (ntaken (the-enat (nlength (nnth nells i))) (nnth nells i))
       using nfilter-nlast[of ((the-enat(nlength (nnth nells i)))) nnth nells i P ]
        using 93 931 932 by force
     have 94: nlast (nfilter P (nnth nells i)) = (nlast (nnth nells i))
       using 932 933 by auto
     have 95: (nlast (nnth nells i)) = (nfirst (nnth nells (Suc i)))
       using a1 assms(4) nlastnfirst-def1 by auto
     show ?thesis
     by (simp add: 91 92 94 95)
    qed
```

```
  qed
 have 10: ¬ nfinite (nmap (nfilter P) nells)
  using assms
  using nfinite-nmap by blast
 have 9: nconcat (nmap (nbutlast) (nmap (nfilter P) nells)) =
       nfusecat (nmap (nfilter P) nells)
  using assms  nfusecat-nconcat-nmap-nbutlast[of (nmap (nfilter P) nells)]
using 10 6 8 9 by presburger

 show ?thesis
 by (simp add: 1 3 4 9)
qed

lemma nsubn-nellist-nfirst-imp-nlast-finite:
 assumes  nidx l
       nfirst l =0
        nfinite σ
       nfinite l
      nlast l = (the-enat(nlength σ))
      ⋀i. i< nlength l ⟹P (nfirst (nnth (nsubn-nellist σ l) i))
       P (nlast (nlast (nsubn-nellist σ l)))
       i < nlength l
 shows P (nlast (nnth (nsubn-nellist σ l) i))
proof −
 have 1: nlength (l) > 0
    using assms(8) by auto
 have 2: (nnth (nsubn-nellist σ l) i) = nsubn σ (nnth l i) (nnth l (Suc i))
  using assms  nnth-nsubn-nellist  by (metis eSuc-enat ileI1)
 have 3: (nlast (nsubn σ (nnth l i) (nnth l (Suc i)))) = (nnth σ (nnth l (Suc i)))
    using assms(1,8) by (metis diff-add eSuc-enat ileI1 nidx-expand nsubn-nlast order-less-le)
 have 4: ⋀i. i< nlength l ⟹P (nnth σ  (nnth l i))
    using assms
    by (metis eSuc-enat ileI1 nnth-nsubn-nellist nsubn-nfirst)
 have 5: ( (nlast (nsubn-nellist σ l))) = (nnth (nsubn-nellist σ l) (the-enat(nlength l) −1))
  using 1 assms(4)
   by (metis One-nat-def idiff-0-right idiff-enat-enat ndropn-Suc-NCons ndropn-all ndropn-nlength nellist-split-2-first nfinite-conv-nlength-enat
    nle-le nlength-nsubn-nellist nnth-NNil nnth-zero-ndropn the-enat.simps zero-enat-def)
 have 6:  (nnth (nsubn-nellist σ l) (the-enat(nlength l) −1)) =
         nsubn σ (nnth l (the-enat(nlength l) −1)) (nnth l (the-enat(nlength l)))
  using assms nnth-nsubn-nellist
   using 1 by (metis One-nat-def Suc-pred nfinite-conv-nlength-enat nle-le not-gr-zero the-enat.simps zero-enat-def)
 have 7:  nlast (nsubn σ (nnth l (the-enat(nlength l) −1)) (nnth l (the-enat(nlength l)))) =
          nnth σ (nnth l (the-enat(nlength l)))
     using 1 assms(1,4)
    by (metis One-nat-def Suc-pred diff-add enat-ord-simps(2) nfinite-conv-nlength-enat nidx-expand nle-le nsubn-nlast order-less-le the-enat.simps
       zero-enat-def)
 have 8:  P (nnth σ (nnth l (Suc i)))
```

using *4 5 6 7 assms*(*7*,*8*) **by** (*metis eSuc-enat ileI1 order-less-le the-enat.simps*)
show *?thesis*
by (*simp add*: *2 3 8*)
qed

lemma *nsubn-nellist-nfirst-imp-nlast*:
assumes *nidx l*
nfirst l =*0*
¬ *nfinite σ*
¬ *nfinite l*
⋀*i*. *P* (*nfirst* (*nnth* (*nsubn-nellist σ l*) *i*))
shows *P* (*nlast* (*nnth* (*nsubn-nellist σ l*) *i*))
using *assms*
by (*metis eSuc-enat enat-ile ileI1 linorder-not-less*
nfinite-conv-nlength-enat nfinite-ntl nlastnfirst-def1
nlastnfirst-nsubn-nellist-infinite nlength-nsubn-nellist
zero-enat-def)

lemma *nfusecat-nfilter-dist-nsubn-nellist-finite*:
assumes *nidx l*
nfirst l =*0*
nfinite σ
nfinite l
nlast l = (*the-enat*(*nlength σ*))
⋀*i*. *i* < *nlength l* ⟹ *P* (*nfirst* (*nnth* (*nsubn-nellist σ l*) *i*))
P (*nlast* (*nlast* (*nsubn-nellist σ l*)))
nlength l > *0*
shows *nfilter P* (*nfusecat* (*nsubn-nellist σ l*)) =
nfusecat (*nmap* (λ*x*. *nfilter P x*) (*nsubn-nellist σ l*))
proof −
have *1*: (⋀*nell*. *nell* ∈ *nset* (*nsubn-nellist σ l*) ⟹ *P* (*nfirst nell*))
using *assms*
by (*metis iless-Suc-eq in-nset-conv-nnth nellist-split-2-first nlength-NCons nlength-nsubn-nellist*)
have *01*: *nlength σ* > *0*
using *assms*
by (*metis enat-ord-simps*(*2*) *nfinite-conv-nlength-enat nfirst-eq-nnth-zero nidx-less-last-1 nnth-nlast the-enat.simps zero-enat-def*)
have *02*: ⋀*x*. *x* ∈ *nset* (*nsubn-nellist σ l*) = (∃ *i*. *i* < *nlength l* ∧ *x* = (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))
using *assms*(*8*)
by (*metis Suc-ile-eq iless-Suc-eq in-nset-conv-nnth nellist-split-2-first nlength-NCons nlength-nsubn-nellist nnth-nsubn-nellist*)
have *2*: (⋀*xx*. *xx* ∈ *nset* (*nsubn-nellist σ l*) ⟹ *0* < *nlength xx*)
using *assms*
using *02*
by (*metis eSuc-enat enat-ord-simps*(*1*) *ileI1 nfinite-conv-nlength-enat nidx-all-le-nlast nidx-expand nsubn-nlength-gr*
the-enat.simps)
have *3*: (⋀*xx*. *xx* ∈ *nset* (*nsubn-nellist σ l*) ⟹ *nfinite xx*)
using *assms*
using *02* **by** (*metis nsubn-nfinite*)

have *4*: *nlastnfirst* (*nsubn-nellist* σ *l*)
using *assms* **using** *nlastnfirst-nsubn-nellist-finite*
by (*metis nfinite-nlength-enat nle-le the-enat.simps*)
have *5*: *nfinite* (*nsubn-nellist* σ *l*)
using *assms*
using *2 3 4* **by** (*metis nfusecat-nfinite nfusecat-nsubn-nellist-finite-var*)
show *?thesis*
using *nfilter-nfusecat-dist-finite*[*of* (*nsubn-nellist* σ *l*) *P*]
using *1 2 3 4 5 assms*(*7*) *nfilter-nfusecat-dist-zero not-gr-zero* **by** *blast*
qed

lemma *nfusecat-nfilter-dist-nsubn-nellist*:
assumes *nidx l*
nfirst l = *0*
¬ *nfinite* σ
¬ *nfinite l*
$\bigwedge i$. *P* (*nfirst* (*nnth* (*nsubn-nellist* σ *l*) *i*))
shows *nfilter P* (*nfusecat* (*nsubn-nellist* σ *l*)) =
nfusecat (*nmap* (λx. *nfilter P x*) (*nsubn-nellist* σ *l*))
proof −
have *1*: ($\bigwedge$*nell*. *nell* ∈ *nset* (*nsubn-nellist* σ *l*) ⟹ *P* (*nfirst nell*))
using *assms* **by** (*metis in-nset-conv-nnth*)
have *2*: ($\bigwedge$*xx*. *xx* ∈ *nset* (*nsubn-nellist* σ *l*) ⟹ *0* < *nlength xx*)
using *assms*
by (*metis Suc-ile-eq enat-ile in-nset-conv-nnth nfinite-conv-nlength-enat nidx-expand nle-le nnth-nsubn-nellist nsubn-nlength-gr-one zero-enat-def*)
have *3*: ($\bigwedge$*xx*. *xx* ∈ *nset* (*nsubn-nellist* σ *l*) ⟹ *nfinite xx*)
using *assms*
by (*metis Suc-ile-eq iless-Suc-eq in-nset-conv-nnth nellist-split-2-first nfinite-conv-nlength-enat nfinite-nnth-nsubn-n nlength-NCons nlength-nsubn-nellist not-gr-zero zero-enat-def*)
have *4*: *nlastnfirst* (*nsubn-nellist* σ *l*)
using *assms* **using** *nlastnfirst-nsubn-nellist-infinite* **by** *auto*
have *5*: ¬ *nfinite* (*nsubn-nellist* σ *l*)
using *assms*
by (*metis 2 3 4 nfusecat-nfinite nfusecat-nsubn-nellist-var*)
show *?thesis*
using *nfilter-nfusecat-dist*[*of* (*nsubn-nellist* σ *l*) *P*]
using *1 2 3 4 5* **by** *blast*
qed

1.3.14 Transfer rule

context includes *lifting-syntax*
begin
lemma *ndropns-transfer2* [*transfer-rule*]:
(*nellist-all2 A* ===> *nellist-all2* (*nellist-all2 A*)) *ndropns ndropns*
unfolding *rel-fun-def*
by (*auto intro*: *nellist-all2-ndropnsI*)

end

end

1.4 Extended Integers

theory *Extended-Int*

imports
HOL−Library.Extended-Nat

begin

datatype *eint* = *eint int* | *PInfty* | *MInfty*

lemma *eint-cong*:
x= *y* ⟹ *eint x* = *eint y*
by *simp*

instantiation *eint* :: *uminus*
begin

fun *uminus-eint* **where**
−(*eint i*) = *eint* (− *i*)
| −*PInfty* = *MInfty*
| −*MInfty* = *PInfty*

instance ..

end

instantiation *eint* :: *infinity*
begin

definition (∞ :: *eint*) = *PInfty*
instance ..

end

declare [[*coercion eint* :: *int* ⇒ *eint*]]

lemma *eint-uminus-uminus*[*simp*]:
fixes *a* :: *eint*
shows − (− *a*) = *a*
by (*cases a*) *simp-all*

lemma
shows *PInfty-eq-infinity*[*simp*]: *PInfty* = ∞
and *MInfty-eq-minfinity*[*simp*]: *MInfty* = − ∞

and *MInfty-neq-PInfty*[*simp*]: $\infty \neq -(\infty::eint) - \infty \neq (\infty::eint)$
and *MInfty-neq-eint*[*simp*]: $eint\ r \neq -\infty - \infty \neq eint\ r$
and *PInfty-neq-eint*[*simp*]: $eint\ r \neq \infty\ \infty \neq eint\ r$
and *PInfty-cases*[*simp*]: $(case\ \infty\ of\ eint\ r \Rightarrow f\ r \mid PInfty \Rightarrow y \mid MInfty \Rightarrow z) = y$
and *MInfty-cases*[*simp*]: $(case - \infty\ of\ eint\ r \Rightarrow f\ r \mid PInfty \Rightarrow y \mid MInfty \Rightarrow z) = z$
by (*simp-all add*: *infinity-eint-def*)

declare
PInfty-eq-infinity[*code-post*]
MInfty-eq-minfinity[*code-post*]

lemma [*code-unfold*]:
$\infty = PInfty$
$-\ PInfty = MInfty$
by *simp-all*

lemma *inj-eint*[*simp*]: *inj-on eint A*
unfolding *inj-on-def* **by** *auto*

lemma *eint-cases*[*cases type*: *eint*]:
obtains (*int*) *i* **where** $x = eint\ i$
| (*PInf*) $x = \infty$
| (*MInf*) $x = -\infty$
by (*cases x*) *auto*

lemmas *eint2-cases* = *eint-cases*[*case-product eint-cases*]
lemmas *eint3-cases* = *eint2-cases*[*case-product eint-cases*]

lemma *eint-all-split*: $\bigwedge P.\ (\forall x::eint.\ P\ x) \longleftrightarrow P\ \infty \wedge (\forall x.\ P\ (eint\ x)) \wedge P\ (-\infty)$
by (*metis eint-cases*)

lemma *eint-ex-split*: $\bigwedge P.\ (\exists x::eint.\ P\ x) \longleftrightarrow P\ \infty \vee (\exists x.\ P\ (eint\ x)) \vee P\ (-\infty)$
by (*metis eint-cases*)

lemma *eint-uminus-eq-iff*[*simp*]:
fixes *a b* :: *eint*
shows $-a = -b \longleftrightarrow a = b$
by (*cases rule*: *eint2-cases*[*of a b*]) *simp-all*

function *int-of-eint* :: *eint* ⇒ *int* **where**
$int\text{-}of\text{-}eint\ (eint\ r) = r$
| $int\text{-}of\text{-}eint\ \infty = 0$
| $int\text{-}of\text{-}eint\ (-\infty) = 0$
by (*auto intro*: *eint-cases*)
termination by *standard* (*rule wf-on-bot*)

lemma *int-of-eint*[*simp*]:
$int\text{-}of\text{-}eint\ (-\ x :: eint) = -\ (int\text{-}of\text{-}eint\ x)$

by (*cases x*) *simp-all*

lemma *range-eint*[*simp*]: *range eint* = *UNIV* − {∞, −∞}
proof *safe*
fix *x*
assume *x* ∉ *range eint x* ≠ ∞
then show *x* = −∞
by (*cases x*) *auto*
qed *auto*

lemma *eint-range-uminus*[*simp*]: *range uminus* = (*UNIV*::*eint set*)
proof *safe*
fix *x* :: *eint*
show *x* ∈ *range uminus*
by (*intro image-eqI*[*of* - - −*x*]) *auto*
qed *auto*

instantiation *eint* :: *abs*
begin

function *abs-eint* **where**
|*eint i*| = *eint* |*i*|
| |−∞| = (∞::*eint*)
| |∞| = (∞::*eint*)
by (*auto intro*: *eint-cases*)
termination proof qed
(*rule wf-on-bot*)

instance ..

end

lemma *abs-eq-infinity-cases*[*elim*!]:
fixes *x* :: *eint*
assumes |*x*| = ∞
obtains *x* = ∞ | *x* = −∞
using *assms* **by** (*cases x*) *auto*

lemma *abs-neq-infinity-cases*[*elim*!]:
fixes *x* :: *eint*
assumes |*x*| ≠ ∞
obtains *r* **where** *x* = *eint r*
using *assms* **by** (*cases x*) *auto*

lemma *abs-eint-uminus*[*simp*]:
fixes *x* :: *eint*
shows |− *x*| = |*x*|
by (*cases x*) *auto*

lemma *eint-infinity-cases*:
fixes a :: *eint*
shows $a \neq \infty \Longrightarrow a \neq -\infty \Longrightarrow |a| \neq \infty$
by *auto*

Addition

instantiation *eint* :: {*one*,*comm-monoid-add*,*zero-neq-one*}
begin

definition *0* = *eint 0*
definition *1* = *eint 1*

function *plus-eint* **where**
eint r + *eint p* = *eint* (*r* + *p*)
| ∞ + *a* = (∞::*eint*)
| *a* + ∞ = (∞::*eint*)
| *eint r* + −∞ = − ∞
| −∞ + *eint p* = −(∞::*eint*)
| −∞ + −∞ = −(∞::*eint*)
proof *goal-cases*
case *prems*: (*1 P x*)
then obtain *a b* **where** *x* = (*a*, *b*)
by (*cases x*) *auto*
with *prems* **show** *P*
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*
qed *auto*
termination by *standard* (*rule wf-on-bot*)

lemma *Infty-neq-0*[*simp*]:
(∞::*eint*) ≠ *0* *0* ≠ (∞::*eint*)
−(∞::*eint*) ≠ *0* *0* ≠ −(∞::*eint*)
by (*simp-all add*: *zero-eint-def*)

lemma *eint-eq-0*[*simp*]:
eint r = *0* ⟷ *r* = *0*
0 = *eint r* ⟷ *r* = *0*
unfolding *zero-eint-def* **by** *simp-all*

lemma *eint-eq-1*[*simp*]:
eint r = *1* ⟷ *r* = *1*
1 = *eint r* ⟷ *r* = *1*
unfolding *one-eint-def* **by** *simp-all*

instance
proof
fix *a b c* :: *eint*

show *0 + a = a*
by (*cases a*) (*simp-all add*: *zero-eint-def*)
show *a + b = b + a*
by (*cases rule*: *eint2-cases*[*of a b*]) *simp-all*
show *a + b + c = a + (b + c)*
by (*cases rule*: *eint3-cases*[*of a b c*]) *simp-all*
show *0 ≠ (1*::*eint)*
by (*simp add*: *one-eint-def zero-eint-def*)
qed

end

lemma *eint-0-plus* [*simp*]: *eint 0 + x = x*
and *plus-eint-0* [*simp*]: *x + eint 0 = x*
by(*simp-all flip*: *zero-eint-def*)

instance *eint* :: *numeral* **..**

lemma *int-of-eint-0*[*simp*]: *int-of-eint* (*0*::*eint*) = *0*
unfolding *zero-eint-def* **by** *simp*

lemma *abs-eint-zero*[*simp*]: |*0*| = (*0*::*eint*)
unfolding *zero-eint-def abs-eint.simps* **by** *simp*

lemma *eint-uminus-zero*[*simp*]: − *0* = (*0*::*eint*)
by (*simp add*: *zero-eint-def*)

lemma *eint-uminus-zero-iff*[*simp*]:
fixes *a* :: *eint*
shows *−a = 0* ⟷ *a = 0*
by (*cases a*) *simp-all*

lemma *eint-plus-eq-PInfty*[*simp*]:
fixes *a b* :: *eint*
shows *a + b* = ∞ ⟷ *a* = ∞ ∨ *b* = ∞
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-plus-eq-MInfty*[*simp*]:
fixes *a b* :: *eint*
shows *a + b* = −∞ ⟷ (*a* = −∞ ∨ *b* = −∞) ∧ *a* ≠ ∞ ∧ *b* ≠ ∞
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-add-cancel-left*:
fixes *a b* :: *eint*
assumes *a* ≠ −∞
shows *a + b = a + c* ⟷ *a* = ∞ ∨ *b = c*
using *assms* **by** (*cases rule*: *eint3-cases*[*of a b c*]) *auto*

lemma *eint-add-cancel-right*:
fixes $a\ b :: eint$
assumes $a \neq -\infty$
shows $b + a = c + a \longleftrightarrow a = \infty \vee b = c$
using *assms* **by** (*cases rule*: *eint3-cases*[*of a b c*]) *auto*

lemma *eint-int*: $eint\ (int\text{-}of\text{-}eint\ x) = (if\ |x| = \infty\ then\ 0\ else\ x)$
by (*cases x*) *simp-all*

lemma *int-of-eint-add*:
fixes $a\ b :: eint$
shows $int\text{-}of\text{-}eint\ (a + b) =$
$(if\ (|a| = \infty) \wedge (|b| = \infty) \vee (|a| \neq \infty) \wedge (|b| \neq \infty)\ then\ int\text{-}of\text{-}eint\ a + int\text{-}of\text{-}eint\ b\ else\ 0)$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

Linear order on *eint*

instantiation *eint* :: *linorder*
begin

function *less-eint*
where
$eint\ x < eint\ y \longleftrightarrow x < y$
$| (\infty::eint) < a \longleftrightarrow False$
$| a < -(\infty::eint) \longleftrightarrow False$
$| eint\ x < \infty \longleftrightarrow True$
$| -\infty < eint\ r \longleftrightarrow True$
$| -\infty < (\infty::eint) \longleftrightarrow True$
proof *goal-cases*
case *prems*: (*1 P x*)
then obtain *a b* **where** $x = (a,b)$ **by** (*cases x*) *auto*
with *prems* **show** *P* **by** (*cases rule*: *eint2-cases*[*of a b*]) *auto*
qed *simp-all*
termination by (*relation* {}) *simp*

definition $x \leq (y::eint) \longleftrightarrow x < y \vee x = y$

lemma *eint-infty-less*[*simp*]:
fixes $x :: eint$
shows $x < \infty \longleftrightarrow (x \neq \infty)$
$-\infty < x \longleftrightarrow (x \neq -\infty)$
by (*cases x*, *simp-all*) (*cases x*, *simp-all*)

lemma *eint-infty-less-eq*[*simp*]:
fixes $x :: eint$
shows $\infty \leq x \longleftrightarrow x = \infty$
and $x \leq -\infty \longleftrightarrow x = -\infty$
by (*auto simp add*: *less-eq-eint-def*)

lemma *eint-less*[*simp*]:

eint $r < 0 \longleftrightarrow (r < 0)$
$0 <$ *eint* $r \longleftrightarrow (0 < r)$
eint $r < 1 \longleftrightarrow (r < 1)$
$1 <$ *eint* $r \longleftrightarrow (1 < r)$
$0 < (\infty :: eint)$
$-(\infty :: eint) < 0$
by (*simp-all add*: *zero-eint-def one-eint-def*)

lemma *eint-less-eq*[*simp*]:
$x \leq (\infty :: eint)$
$-(\infty :: eint) \leq x$
eint $r \leq$ *eint* $p \longleftrightarrow r \leq p$
eint $r \leq 0 \longleftrightarrow r \leq 0$
$0 \leq$ *eint* $r \longleftrightarrow 0 \leq r$
eint $r \leq 1 \longleftrightarrow r \leq 1$
$1 \leq$ *eint* $r \longleftrightarrow 1 \leq r$
by (*auto simp add*: *less-eq-eint-def zero-eint-def one-eint-def*)

lemma *eint-infty-less-eq2*:
$a \leq b \Longrightarrow a = \infty \Longrightarrow b = (\infty :: eint)$
$a \leq b \Longrightarrow b = -\infty \Longrightarrow a = -(\infty :: eint)$
by *simp-all*

instance
proof
fix $x\ y\ z :: eint$
show $x \leq x$
by (*cases x*) *simp-all*
show $x < y \longleftrightarrow x \leq y \wedge \neg\ y \leq x$
by (*cases rule*: *eint2-cases*[*of x y*]) *auto*
show $x \leq y \vee y \leq x$
by (*cases rule*: *eint2-cases*[*of x y*]) *auto*
{
assume $x \leq y\ y \leq x$
then show $x = y$
by (*cases rule*: *eint2-cases*[*of x y*]) *auto*
}
{
assume $x \leq y\ y \leq z$
then show $x \leq z$
by (*cases rule*: *eint3-cases*[*of x y z*]) *auto*
}
qed

end

instance *eint* :: *ordered-comm-monoid-add*
proof
fix $a\ b\ c :: eint$
assume $a \leq b$

then show $c + a \leq c + b$
by (*cases rule*: *eint3-cases*[*of a b c*]) *auto*
qed

lemma *eint-one-not-less-zero-eint*[*simp*]: $\neg\ 1 < (0::eint)$
by (*simp add*: *zero-eint-def*)

lemma *int-of-eint-positive-mono*:
fixes $x\ y :: eint$
shows $0 \leq x \Longrightarrow x \leq y \Longrightarrow y \neq \infty \Longrightarrow int\text{-}of\text{-}eint\ x \leq int\text{-}of\text{-}eint\ y$
by (*cases rule*: *eint2-cases*[*of x y*]) *auto*

lemma *eint-MInfty-lessI*[*intro*, *simp*]:
fixes $a :: eint$
shows $a \neq -\infty \Longrightarrow -\infty < a$
by (*cases a*) *auto*

lemma *eint-less-PInfty*[*intro*, *simp*]:
fixes $a :: eint$
shows $a \neq \infty \Longrightarrow a < \infty$
by (*cases a*) *auto*

lemma *eint-less-eint-Ex*:
fixes $a\ b :: eint$
shows $x < eint\ r \longleftrightarrow x = -\infty \vee (\exists\, p.\ p < r \wedge x = eint\ p)$
by (*cases x*) *auto*

lemma *less-PInf-Ex-of-nat*: $x \neq \infty \longleftrightarrow (\exists\, n::nat.\ x < eint\ (int\ n))$
proof (*cases x*)
case (*int r*)
then show *?thesis*
by (*metis gt-ex int-ops*(*1*) *less-eint.simps*(*1*) *less-eint.simps*(*2*) *linorder-less-linear of-nat-less-iff zero-less-imp-eq-int*)
qed *simp-all*

lemma *eint-add-strict-mono2*:
fixes $a\ b\ c\ d :: eint$
assumes $a < b$ **and** $c < d$
shows $a + c < b + d$
using *assms*
by (*cases a*; *force simp add*: *elim*: *less-eint.elims*)

lemma *eint-minus-le-minus*[*simp*]:
fixes $a\ b :: eint$
shows $-\ a \leq -\ b \longleftrightarrow b \leq a$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-minus-less-minus*[*simp*]:
fixes $a\ b :: eint$
shows $-\ a < -\ b \longleftrightarrow b < a$

by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-le-int-iff*:
$x \leq$ *int-of-eint* $y \longleftrightarrow (|y| \neq \infty \longrightarrow$ *eint* $x \leq y) \wedge (|y| = \infty \longrightarrow x \leq 0)$
by (*cases y*) *auto*

lemma *int-le-eint-iff*:
int-of-eint $y \leq x \longleftrightarrow (|y| \neq \infty \longrightarrow y \leq$ *eint* $x) \wedge (|y| = \infty \longrightarrow 0 \leq x)$
by (*cases y*) *auto*

lemma *eint-less-int-iff*:
$x <$ *int-of-eint* $y \longleftrightarrow (|y| \neq \infty \longrightarrow$ *eint* $x < y) \wedge (|y| = \infty \longrightarrow x < 0)$
by (*cases y*) *auto*

lemma *int-less-eint-iff*:
int-of-eint $y < x \longleftrightarrow (|y| \neq \infty \longrightarrow y <$ *eint* $x) \wedge (|y| = \infty \longrightarrow 0 < x)$
by (*cases y*) *auto*

To help with inferences like $[\![a <$ *eint* $x; x < y]\!] \Longrightarrow a <$ *eint* y, where x and y are int.

lemma *le-eint-le*: $a \leq$ *eint* $x \Longrightarrow x \leq y \Longrightarrow a \leq$ *eint* y
using *eint-less-eq*(*3*) *order.trans* **by** *blast*

lemma *le-eint-less*: $a \leq$ *eint* $x \Longrightarrow x < y \Longrightarrow a <$ *eint* y
by (*simp add*: *le-less-trans*)

lemma *less-eint-le*: $a <$ *eint* $x \Longrightarrow x \leq y \Longrightarrow a <$ *eint* y
using *eint-less-eint-Ex* **by** *auto*

lemma *eint-le-le*: *eint* $y \leq a \Longrightarrow x \leq y \Longrightarrow$ *eint* $x \leq a$
by (*simp add*: *order-subst2*)

lemma *eint-le-less*: *eint* $y \leq a \Longrightarrow x < y \Longrightarrow$ *eint* $x < a$
by (*simp add*: *dual-order.strict-trans1*)

lemma *eint-less-le*: *eint* $y < a \Longrightarrow x \leq y \Longrightarrow$ *eint* $x < a$
using *eint-less-eq*(*3*) *le-less-trans* **by** *blast*

lemma *int-of-eint-pos*:
fixes $x ::$ *eint*
shows $0 \leq x \Longrightarrow 0 \leq$ *int-of-eint* x **by** (*cases x*) *auto*

lemmas *int-of-eint-ord-simps* =
eint-le-int-iff int-le-eint-iff eint-less-int-iff int-less-eint-iff

lemma *abs-eint-ge0*[*simp*]: $0 \leq x \Longrightarrow |x :: eint| = x$
by (*cases x*) *auto*

lemma *abs-eint-less0*[*simp*]: $x < 0 \Longrightarrow |x :: eint| = -x$
by (*cases x*) *auto*

lemma *abs-eint-pos*[*simp*]: $0 \leq |x :: eint|$
by (*cases x*) *auto*

lemma *eint-abs-leI*:
fixes $x\ y :: eint$
shows $[\![x \leq y; -x \leq y]\!] \Longrightarrow |x| \leq y$
by(*cases x y rule*: *eint2-cases*)(*simp-all*)

lemma *eint-abs-add*:
fixes *a b*::*eint*
shows *abs*(*a*+*b*) ≤ *abs a* + *abs b*
by (*cases rule*: *eint2-cases*[*of a b*]) (*auto*)

lemma *int-of-eint-le-0*[*simp*]: *int-of-eint* $(x :: eint) \leq 0 \longleftrightarrow x \leq 0 \vee x = \infty$
by (*cases x*) *auto*

lemma *abs-int-of-eint*[*simp*]: $|$*int-of-eint* $(x :: eint)| =$ *int-of-eint* $|x|$
by (*cases x*) *auto*

lemma *zero-less-int-of-eint*:
fixes $x :: eint$
shows $0 <$ *int-of-eint* $x \longleftrightarrow 0 < x \wedge x \neq \infty$
by (*cases x*) *auto*

lemma *eint-0-le-uminus-iff*[*simp*]:
fixes $a :: eint$
shows $0 \leq -\ a \longleftrightarrow a \leq 0$
by (*cases rule*: *eint2-cases*[*of a*]) *auto*

lemma *eint-uminus-le-0-iff*[*simp*]:
fixes $a :: eint$
shows $-\ a \leq 0 \longleftrightarrow 0 \leq a$
by (*cases rule*: *eint2-cases*[*of a*]) *auto*

lemma *eint-add-strict-mono*:
fixes $a\ b\ c\ d :: eint$
assumes $a \leq b$
and $0 \leq a$
and $a \neq \infty$
and $c < d$
shows $a + c < b + d$
using *assms*
by (*cases rule*: *eint3-cases*[*case-product eint-cases*, *of a b c d*]) *auto*

lemma *eint-less-add*:
fixes $a\ b\ c :: eint$
shows $|a| \neq \infty \Longrightarrow c < b \Longrightarrow a + c < a + b$
by (*cases rule*: *eint2-cases*[*of b c*]) *auto*

lemma *eint-add-nonneg-eq-0-iff*:

fixes a b :: *eint*
shows $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow a + b = 0 \longleftrightarrow a = 0 \wedge b = 0$
by (*cases a b rule*: *eint2-cases*) *auto*

lemma *eint-uminus-eq-reorder*: $- a = b \longleftrightarrow a = (-b::eint)$
by *auto*

lemma *eint-uminus-less-reorder*: $- a < b \longleftrightarrow -b < (a::eint)$
by (*subst* (*3*) *eint-uminus-uminus*[*symmetric*]) (*simp only*: *eint-minus-less-minus*)

lemma *eint-less-uminus-reorder*: $a < - b \longleftrightarrow b < - (a::eint)$
by (*subst* (*3*) *eint-uminus-uminus*[*symmetric*]) (*simp only*: *eint-minus-less-minus*)

lemma *eint-uminus-le-reorder*: $- a \leq b \longleftrightarrow -b \leq (a::eint)$
by (*subst* (*3*) *eint-uminus-uminus*[*symmetric*]) (*simp only*: *eint-minus-le-minus*)

lemmas *eint-uminus-reorder* =
eint-uminus-eq-reorder eint-uminus-less-reorder eint-uminus-le-reorder

lemma *eint-bot*:
fixes x :: *eint*
assumes $\bigwedge B.\ x \leq eint\ B$
shows $x = - \infty$
proof (*cases x*)
case (*int r*)
with *assms*[*of r − 1*] **show** *?thesis*
by *auto*
next
case *PInf*
with *assms*[*of 0*] **show** *?thesis*
by *auto*
next
case *MInf*
then show *?thesis*
by *simp*
qed

lemma *eint-top*:
fixes x :: *eint*
assumes $\bigwedge B.\ x \geq eint\ B$
shows $x = \infty$
proof (*cases x*)
case (*int r*)
with *assms*[*of r + 1*] **show** *?thesis*
by *auto*
next
case *MInf*
with *assms*[*of 0*] **show** *?thesis*
by *auto*
next

```
  case PInf
  then show ?thesis
    by simp
qed

lemma
  shows eint-max[simp]: eint (max x y) = max (eint x) (eint y)
    and eint-min[simp]: eint (min x y) = min (eint x) (eint y)
  by (simp-all add: min-def max-def)

lemma eint-max-0: max 0 (eint r) = eint (max 0 r)
  by (auto simp: zero-eint-def)

lemma
  fixes f :: nat ⇒ eint
  shows eint-incseq-uminus[simp]: incseq (λx. − f x) ⟷ decseq f
    and eint-decseq-uminus[simp]: decseq (λx. − f x) ⟷ incseq f
  unfolding decseq-def incseq-def by auto

lemma incseq-eint: incseq f ⟹ incseq (λx. eint (f x))
  unfolding incseq-def by auto

lemma sum-eint[simp]: (∑ x∈A. eint (f x)) = eint (∑ x∈A. f x)
proof (cases finite A)
  case True
  then show ?thesis by induct auto
next
  case False
  then show ?thesis by simp
qed

lemma sum-list-eint [simp]: sum-list (map (λx. eint (f x)) xs) = eint (sum-list (map f xs))
  by (induction xs) simp-all

lemma sum-Pinfty:
  fixes f :: 'a ⇒ eint
  shows (∑ x∈P. f x) = ∞ ⟷ finite P ∧ (∃ i∈P. f i = ∞)
proof safe
  assume *: sum f P = ∞
  show finite P
  proof (rule ccontr)
    assume ¬ finite P
    with * show False
      by auto
  qed
  show ∃ i∈P. f i = ∞
  proof (rule ccontr)
    assume ¬ ?thesis
    then have ⋀i. i ∈ P ⟹ f i ≠ ∞
      by auto
```

```
    with ‹finite P› have sum f P ≠ ∞
      by induct auto
    with ∗ show False
      by auto
  qed
next
  fix i
  assume finite P and i ∈ P and f i = ∞
  then show sum f P = ∞
  proof induct
    case (insert x A)
    show ?case using insert by (cases x = i) auto
  qed simp
qed

lemma sum-Inf:
  fixes f :: 'a ⇒ eint
  shows |sum f A| = ∞ ⟷ finite A ∧ (∃ i∈A. |f i| = ∞)
proof
  assume ∗: |sum f A| = ∞
  have finite A
    by (rule ccontr) (insert ∗, auto)
  moreover have ∃ i∈A. |f i| = ∞
  proof (rule ccontr)
    assume ¬ ?thesis
    then have ∀ i∈A. ∃ r. f i = eint r
      by auto
    from bchoice[OF this] obtain r where ∀ x∈A. f x = eint (r x) ..
    with ∗ show False
      by auto
  qed
  ultimately show finite A ∧ (∃ i∈A. |f i| = ∞)
    by auto
next
  assume finite A ∧ (∃ i∈A. |f i| = ∞)
  then obtain i where finite A i ∈ A and |f i| = ∞
    by auto
  then show |sum f A| = ∞
  proof induct
    case (insert j A)
    then show ?case
      by (cases rule: eint3-cases[of f i f j sum f A]) auto
  qed simp
qed

lemma sum-int-of-eint:
  fixes f :: 'i ⇒ eint
  assumes ⋀x. x ∈ S ⟹ |f x| ≠ ∞
  shows (∑ x∈S. int-of-eint (f x)) = int-of-eint (sum f S)
proof −
```

```
  have ∀ x∈S. ∃ r. f x = eint r
  proof
    fix x
    assume x ∈ S
    from assms[OF this] show ∃ r. f x = eint r
      by (cases f x) auto
  qed
  from bchoice[OF this] obtain r where ∀ x∈S. f x = eint (r x) ..
  then show ?thesis
    by simp
qed

lemma sum-eint-0:
  fixes f :: 'a ⇒ eint
  assumes finite A
    and ⋀i. i ∈ A ⟹ 0 ≤ f i
  shows (∑ x∈A. f x) = 0 ⟷ (∀ i∈A. f i = 0)
proof
  assume sum f A = 0 with assms show ∀ i∈A. f i = 0
  proof (induction A)
    case (insert a A)
    then have f a = 0 ∧ (∑ a∈A. f a) = 0
      by (subst eint-add-nonneg-eq-0-iff[symmetric]) (simp-all add: sum-nonneg)
    with insert show ?case
      by simp
  qed simp
qed auto
```

Multiplication

```
instantiation eint :: {comm-monoid-mult,sgn}
begin

function sgn-eint :: eint ⇒ eint where
  sgn (eint r) = eint (sgn r)
| sgn (∞::eint) = 1
| sgn (−∞::eint) = −1
by (auto intro: eint-cases)
termination by standard (rule wf-on-bot)

function times-eint where
  eint r * eint p = eint (r * p)
| eint r * ∞ = (if r = 0 then 0 else if r > 0 then ∞ else −∞)
| ∞ * eint r = (if r = 0 then 0 else if r > 0 then ∞ else −∞)
| eint r * −∞ = (if r = 0 then 0 else if r > 0 then −∞ else ∞)
| −∞ * eint r = (if r = 0 then 0 else if r > 0 then −∞ else ∞)
| (∞::eint) * ∞ = ∞
| −(∞::eint) * ∞ = −∞
| (∞::eint) * −∞ = −∞
| −(∞::eint) * −∞ = ∞
```

proof *goal-cases*
  case *prems*: (*1 P x*)
  then obtain *a b* **where** *x* = (*a*, *b*)
    by (*cases x*) *auto*
  with *prems* **show** *P*
    by (*cases rule*: *eint2-cases*[*of a b*]) *auto*
qed *simp-all*
termination by (*relation* {}) *simp*

instance
proof
  fix *a b c* :: *eint*
  show *1* ∗ *a* = *a*
    by (*cases a*) (*simp-all add*: *one-eint-def*)
  show *a* ∗ *b* = *b* ∗ *a*
    by (*cases rule*: *eint2-cases*[*of a b*]) *simp-all*
  show *a* ∗ *b* ∗ *c* = *a* ∗ (*b* ∗ *c*)
    by (*cases rule*: *eint3-cases*[*of a b c*])
      (*simp-all add*: *zero-eint-def zero-less-mult-iff*)
qed

end

lemma [*simp*]:
  shows *eint-1-times*: *eint 1* ∗ *x* = *x*
  and *times-eint-1*: *x* ∗ *eint 1* = *x*
by(*simp-all flip*: *one-eint-def*)

lemma *one-not-le-zero-eint*[*simp*]: ¬ (*1* ≤ (*0*::*eint*))
  by (*simp add*: *one-eint-def zero-eint-def*)

lemma *int-eint-1*[*simp*]: *int-of-eint* (*1*::*eint*) = *1*
  unfolding *one-eint-def* **by** *simp*

lemma *int-of-eint-le-1*:
  fixes *a* :: *eint*
  shows *a* ≤ *1* ⟹ *int-of-eint a* ≤ *1*
  by (*cases a*) (*auto simp*: *one-eint-def*)

lemma *abs-eint-one*[*simp*]: |*1*| = (*1*::*eint*)
  unfolding *one-eint-def* **by** *simp*

lemma *eint-mult-zero*[*simp*]:
  fixes *a* :: *eint*
  shows *a* ∗ *0* = *0*
  by (*cases a*) (*simp-all add*: *zero-eint-def*)

lemma *eint-zero-mult*[*simp*]:
  fixes *a* :: *eint*
  shows *0* ∗ *a* = *0*

by (*cases a*) (*simp-all add*: *zero-eint-def*)

lemma *eint-m1-less-0*[*simp*]: $-(1::eint) < 0$
by (*simp add*: *zero-eint-def one-eint-def*)

lemma *eint-times*[*simp*]:
$1 \neq (\infty::eint)$ $(\infty::eint) \neq 1$
$1 \neq -(\infty::eint)$ $-(\infty::eint) \neq 1$
by (*auto simp*: *one-eint-def*)

lemma *eint-plus-1*[*simp*]:
$1 + eint\ r = eint\ (r + 1)$
$eint\ r + 1 = eint\ (r + 1)$
$1 + -(\infty::eint) = -\infty$
$-(\infty::eint) + 1 = -\infty$
unfolding *one-eint-def* **by** *auto*

lemma *eint-zero-times*[*simp*]:
fixes $a\ b :: eint$
shows $a * b = 0 \longleftrightarrow a = 0 \vee b = 0$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-mult-eq-PInfty*[*simp*]:
$a * b = (\infty::eint) \longleftrightarrow$
$(a = \infty \wedge b > 0) \vee (a > 0 \wedge b = \infty) \vee (a = -\infty \wedge b < 0) \vee (a < 0 \wedge b = -\infty)$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-mult-eq-MInfty*[*simp*]:
$a * b = -(\infty::eint) \longleftrightarrow$
$(a = \infty \wedge b < 0) \vee (a < 0 \wedge b = \infty) \vee (a = -\infty \wedge b > 0) \vee (a > 0 \wedge b = -\infty)$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-abs-mult*: $|x * y :: eint| = |x| * |y|$
by (*cases x y rule*: *eint2-cases*) (*auto simp*: *abs-mult*)

lemma *eint-0-less-1*[*simp*]: $0 < (1::eint)$
by (*simp-all add*: *zero-eint-def one-eint-def*)

lemma *eint-mult-minus-left*[*simp*]:
fixes $a\ b :: eint$
shows $-a * b = -(a * b)$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-mult-minus-right*[*simp*]:
fixes $a\ b :: eint$
shows $a * -b = -(a * b)$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-mult-infty*[*simp*]:
$a * (\infty::eint) = (if\ a = 0\ then\ 0\ else\ if\ 0 < a\ then\ \infty\ else\ -\infty)$

by (*cases a*) *auto*

lemma *eint-infty-mult*[*simp*]:
(∞::*eint*) * *a* = (*if a* = *0 then 0 else if 0* < *a then* ∞ *else* − ∞)
by (*cases a*) *auto*

lemma *eint-mult-strict-right-mono*:
assumes *a* < *b*
and *0* < *c*
and *c* < (∞::*eint*)
shows *a* * *c* < *b* * *c*
using *assms*
by (*cases rule*: *eint3-cases*[*of a b c*]) (*auto simp*: *zero-le-mult-iff*)

lemma *eint-mult-strict-left-mono*:
a < *b* ⟹ *0* < *c* ⟹ *c* < (∞::*eint*) ⟹ *c* * *a* < *c* * *b*
using *eint-mult-strict-right-mono*
by (*simp add*: *mult.commute*[*of c*])

lemma *eint-mult-right-mono*:
fixes *a b c* :: *eint*
assumes *a* ≤ *b 0* ≤ *c*
shows *a* * *c* ≤ *b* * *c*
proof (*cases c* = *0*)
case *False*
with *assms* **show** *?thesis*
by (*cases rule*: *eint3-cases*[*of a b c*]) *auto*
qed *auto*

lemma *eint-mult-left-mono*:
fixes *a b c* :: *eint*
shows *a* ≤ *b* ⟹ *0* ≤ *c* ⟹ *c* * *a* ≤ *c* * *b*
using *eint-mult-right-mono*
by (*simp add*: *mult.commute*[*of c*])

lemma *eint-mult-mono*:
fixes *a b c d*::*eint*
assumes *b* ≥ *0 c* ≥ *0 a* ≤ *b c* ≤ *d*
shows *a* * *c* ≤ *b* * *d*
by (*metis eint-mult-right-mono mult.commute order-trans assms*)

lemma *eint-mult-mono′*:
fixes *a b c d*::*eint*
assumes *a* ≥ *0 c* ≥ *0 a* ≤ *b c* ≤ *d*
shows *a* * *c* ≤ *b* * *d*
by (*metis eint-mult-right-mono mult.commute order-trans assms*)

lemma *eint-mult-mono-strict*:
fixes *a b c d*::*eint*
assumes *b* > *0 c* > *0 a* < *b c* < *d*

shows $a * c < b * d$
proof −
have $c < \infty$ **using** ‹$c < d$› **by** *auto*
then have $a * c < b * c$ **by** (*metis eint-mult-strict-left-mono*[*OF assms*(*3*) *assms*(*2*)] *mult.commute*)
moreover have $b * c \leq b * d$ **using** *assms*(*2*) *assms*(*4*) **by** (*simp add*: *assms*(*1*) *eint-mult-left-mono less-imp-le*)
ultimately show *?thesis* **by** *simp*
qed

lemma *eint-mult-mono-strict'*:
fixes $a\ b\ c\ d$::*eint*
assumes $a > 0\ c > 0\ a < b\ c < d$
shows $a * c < b * d$
using *assms eint-mult-mono-strict* **by** *auto*

lemma *zero-less-one-eint*[*simp*]: $0 \leq (1$::*eint*)
by (*simp add*: *one-eint-def zero-eint-def*)

lemma *eint-0-le-mult*[*simp*]: $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow 0 \leq a * (b :: eint)$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-right-distrib*:
fixes $r\ a\ b :: eint$
shows $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow r * (a + b) = r * a + r * b$
by (*cases rule*: *eint3-cases*[*of r a b*]) (*simp-all add*: *field-simps*)

lemma *eint-left-distrib*:
fixes $r\ a\ b :: eint$
shows $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow (a + b) * r = a * r + b * r$
by (*cases rule*: *eint3-cases*[*of r a b*]) (*simp-all add*: *field-simps*)

lemma *eint-mult-le-0-iff*:
fixes $a\ b :: eint$
shows $a * b \leq 0 \longleftrightarrow (0 \leq a \wedge b \leq 0) \vee (a \leq 0 \wedge 0 \leq b)$
by (*cases rule*: *eint2-cases*[*of a b*]) (*simp-all add*: *mult-le-0-iff*)

lemma *eint-zero-le-0-iff*:
fixes $a\ b :: eint$
shows $0 \leq a * b \longleftrightarrow (0 \leq a \wedge 0 \leq b) \vee (a \leq 0 \wedge b \leq 0)$
by (*cases rule*: *eint2-cases*[*of a b*]) (*simp-all add*: *zero-le-mult-iff*)

lemma *eint-mult-less-0-iff*:
fixes $a\ b :: eint$
shows $a * b < 0 \longleftrightarrow (0 < a \wedge b < 0) \vee (a < 0 \wedge 0 < b)$
by (*cases rule*: *eint2-cases*[*of a b*]) (*simp-all add*: *mult-less-0-iff*)

lemma *eint-zero-less-0-iff*:
fixes $a\ b :: eint$
shows $0 < a * b \longleftrightarrow (0 < a \wedge 0 < b) \vee (a < 0 \wedge b < 0)$
by (*cases rule*: *eint2-cases*[*of a b*]) (*simp-all add*: *zero-less-mult-iff*)

lemma *eint-left-mult-cong*:
fixes $a\ b\ c :: eint$
shows $c = d \Longrightarrow (d \neq 0 \Longrightarrow a = b) \Longrightarrow a * c = b * d$
by (*cases* $c = 0$) *simp-all*

lemma *eint-right-mult-cong*:
fixes $a\ b\ c :: eint$
shows $c = d \Longrightarrow (d \neq 0 \Longrightarrow a = b) \Longrightarrow c * a = d * b$
by (*cases* $c = 0$) *simp-all*

lemma *eint-distrib*:
fixes $a\ b\ c :: eint$
assumes $a \neq \infty \vee b \neq -\infty$
and $a \neq -\infty \vee b \neq \infty$
and $|c| \neq \infty$
shows $(a + b) * c = a * c + b * c$
using *assms*
by (*cases rule*: *eint3-cases*[*of a b c*]) (*simp-all add*: *field-simps*)

lemma *numeral-eq-eint* [*simp*]: $numeral\ w = eint\ (numeral\ w)$
proof (*induct w rule*: *num-induct*)
case *One*
then show *?case*
by *simp*
next
case (*inc x*)
then show *?case*
by (*simp add*: *inc numeral-inc*)
qed

lemma *distrib-left-eint-nn*:
$c \geq 0 \Longrightarrow (x + y) * eint\ c = x * eint\ c + y * eint\ c$
by(*cases x y rule*: *eint2-cases*)(*simp-all add*: *ring-distribs*)

lemma *sum-eint-right-distrib*:
fixes $f :: 'a \Rightarrow eint$
shows $(\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i) \Longrightarrow r * sum\ f\ A = (\sum n{\in}A.\ r * f\ n)$
by (*induct A rule*: *infinite-finite-induct*) (*auto simp*: *eint-right-distrib sum-nonneg*)

lemma *sum-eint-left-distrib*:
$(\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i) \Longrightarrow sum\ f\ A * r = (\sum n{\in}A.\ f\ n * r :: eint)$
using *sum-eint-right-distrib*[*of A f r*] **by** (*simp add*: *mult-ac*)

lemma *sum-distrib-right-eint*:
$c \geq 0 \Longrightarrow sum\ f\ A * eint\ c = (\sum x{\in}A.\ f\ x * c :: eint)$
by(*subst sum-comp-morphism*[**where** $h{=}\lambda x.\ x * eint\ c$, *symmetric*])(*simp-all add*: *distrib-left-eint-nn*)

lemma *eint-le-int*:
fixes $x\ y :: eint$
assumes $\bigwedge z.\ x \leq eint\ z \Longrightarrow y \leq eint\ z$
shows $y \leq x$
by (*metis assms eint-bot eint-cases eint-infty-less-eq*(*2*) *eint-less-eq*(*1*) *linorder-le-cases*)

lemma *prod-eint-0*:
fixes $f :: 'a \Rightarrow eint$
shows $(\prod i{\in}A.\ f\ i) = 0 \longleftrightarrow finite\ A \wedge (\exists\, i{\in}A.\ f\ i = 0)$
proof (*cases finite A*)
case *True*
then show *?thesis* **by** (*induct A*) *auto*
qed *auto*

lemma *prod-eint-pos*:
fixes $f :: 'a \Rightarrow eint$
assumes *pos*: $\bigwedge i.\ i \in I \Longrightarrow 0 \leq f\ i$
shows $0 \leq (\prod i{\in}I.\ f\ i)$
proof (*cases finite I*)
case *True*
from *this pos* **show** *?thesis*
by *induct auto*
qed *auto*

lemma *prod-PInf*:
fixes $f :: 'a \Rightarrow eint$
assumes $\bigwedge i.\ i \in I \Longrightarrow 0 \leq f\ i$
shows $(\prod i{\in}I.\ f\ i) = \infty \longleftrightarrow finite\ I \wedge (\exists\, i{\in}I.\ f\ i = \infty) \wedge (\forall\, i{\in}I.\ f\ i \neq 0)$
proof (*cases finite I*)
case *True*
from *this assms* **show** *?thesis*
proof (*induct I*)
case (*insert i I*)
then have *pos*: $0 \leq f\ i\ 0 \leq prod\ f\ I$
by (*auto intro*!: *prod-eint-pos*)
from *insert* **have** $(\prod j{\in}insert\ i\ I.\ f\ j) = \infty \longleftrightarrow prod\ f\ I * f\ i = \infty$
by *auto*
also have $\ldots \longleftrightarrow (prod\ f\ I = \infty \vee f\ i = \infty) \wedge f\ i \neq 0 \wedge prod\ f\ I \neq 0$
using *prod-eint-pos*[*of I f*] *pos*
by (*cases rule*: *eint2-cases*[*of f i prod f I*]) *auto*
also have $\ldots \longleftrightarrow finite\ (insert\ i\ I) \wedge (\exists\, j{\in}insert\ i\ I.\ f\ j = \infty) \wedge (\forall\, j{\in}insert\ i\ I.\ f\ j \neq 0)$
using *insert* **by** (*auto simp*: *prod-eint-0*)
finally show *?case* **.**
qed *simp*
qed *auto*

lemma *prod-eint*: $(\prod i{\in}A.\ eint\ (f\ i)) = eint\ (prod\ f\ A)$

proof (*cases finite A*)
  case *True*
  then show *?thesis*
    by *induct* (*auto simp*: *one-eint-def*)
next
  case *False*
  then show *?thesis*
    by (*simp add*: *one-eint-def*)
qed

Power

lemma *eint-power*[*simp*]: (*eint x*) ^ *n* = *eint* (*x*^*n*)
  by (*induct n*) (*auto simp*: *one-eint-def*)

lemma *eint-power-PInf*[*simp*]: (∞::*eint*) ^ *n* = (*if n* = *0 then 1 else* ∞)
  by (*induct n*) (*auto simp*: *one-eint-def*)

lemma *eint-power-uminus*[*simp*]:
  fixes *x* :: *eint*
  shows (− *x*) ^ *n* = (*if even n then x* ^ *n else* − (*x*^*n*))
  by (*induct n*) (*auto simp*: *one-eint-def*)

lemma *eint-power-numeral*[*simp*]:
  (*numeral num* :: *eint*) ^ *n* = *eint* (*numeral num* ^ *n*)
  by (*induct n*) (*auto simp*: *one-eint-def*)

lemma *zero-le-power-eint*[*simp*]:
  fixes *a* :: *eint*
  assumes *0* ≤ *a*
  shows *0* ≤ *a* ^ *n*
  using *assms* **by** (*induct n*) (*auto simp*: *eint-zero-le-0-iff*)

Subtraction

lemma *eint-minus-minus-image*[*simp*]:
  fixes *S* :: *eint set*
  shows *uminus* ‘ *uminus* ‘ *S* = *S*
  by (*auto simp*: *image-iff*)

lemma *eint-uminus-lessThan*[*simp*]:
  fixes *a* :: *eint*
  shows *uminus* ‘ {..<*a*} = {−*a*<..}
proof −
  {
    fix *x*
    assume −*a* < *x*
    then have − *x* < − (− *a*)
      by (*simp del*: *eint-uminus-uminus*)
    then have − *x* < *a*
      by *simp*

}
then show *?thesis*
by *force*
qed

lemma *eint-uminus-greaterThan*[*simp*]: *uminus* ‘ {(*a*::*eint*)<..} = {..<−*a*}
by (*metis eint-uminus-lessThan eint-uminus-uminus eint-minus-minus-image*)

instantiation *eint* :: *minus*
begin

definition $x - y = x + -(y::eint)$
instance ..

end

lemma *eint-minus*[*simp*]:
$eint\ r - eint\ p = eint\ (r - p)$
$-\infty - eint\ r = -\infty$
$eint\ r - \infty = -\infty$
$(\infty::eint) - x = \infty$
$-(\infty::eint) - \infty = -\infty$
$x - -y = x + y$
$x - 0 = x$
$0 - x = -x$
by (*simp-all add*: *minus-eint-def*)

lemma *eint-x-minus-x*[*simp*]: $x - x = (if\ |x| = \infty\ then\ \infty\ else\ 0::eint)$
by (*cases x*) *simp-all*

lemma *eint-eq-minus-iff*:
fixes *x y z* :: *eint*
shows $x = z - y \longleftrightarrow$
$(|y| \neq \infty \longrightarrow x + y = z) \wedge$
$(y = -\infty \longrightarrow x = \infty) \wedge$
$(y = \infty \longrightarrow z = \infty \longrightarrow x = \infty) \wedge$
$(y = \infty \longrightarrow z \neq \infty \longrightarrow x = -\infty)$
by (*cases rule*: *eint3-cases*[*of x y z*]) *auto*

lemma *eint-eq-minus*:
fixes *x y z* :: *eint*
shows $|y| \neq \infty \Longrightarrow x = z - y \longleftrightarrow x + y = z$
by (*auto simp*: *eint-eq-minus-iff*)

lemma *eint-less-minus-iff*:
fixes *x y z* :: *eint*
shows $x < z - y \longleftrightarrow$
$(y = \infty \longrightarrow z = \infty \wedge x \neq \infty) \wedge$
$(y = -\infty \longrightarrow x \neq \infty) \wedge$
$(|y| \neq \infty \longrightarrow x + y < z)$

by (*cases rule*: *eint3-cases*[*of x y z*]) *auto*

lemma *eint-less-minus*:
fixes $x\ y\ z$ :: *eint*
shows $|y| \neq \infty \Longrightarrow x < z - y \longleftrightarrow x + y < z$
by (*auto simp*: *eint-less-minus-iff*)

lemma *eint-le-minus-iff*:
fixes $x\ y\ z$ :: *eint*
shows $x \leq z - y \longleftrightarrow (y = \infty \longrightarrow z \neq \infty \longrightarrow x = -\infty) \wedge (|y| \neq \infty \longrightarrow x + y \leq z)$
by (*cases rule*: *eint3-cases*[*of x y z*]) *auto*

lemma *eint-le-minus*:
fixes $x\ y\ z$ :: *eint*
shows $|y| \neq \infty \Longrightarrow x \leq z - y \longleftrightarrow x + y \leq z$
by (*auto simp*: *eint-le-minus-iff*)

lemma *eint-minus-less-iff*:
fixes $x\ y\ z$ :: *eint*
shows $x - y < z \longleftrightarrow y \neq -\infty \wedge (y = \infty \longrightarrow x \neq \infty \wedge z \neq -\infty) \wedge (y \neq \infty \longrightarrow x < z + y)$
by (*cases rule*: *eint3-cases*[*of x y z*]) *auto*

lemma *eint-minus-less*:
fixes $x\ y\ z$ :: *eint*
shows $|y| \neq \infty \Longrightarrow x - y < z \longleftrightarrow x < z + y$
by (*auto simp*: *eint-minus-less-iff*)

lemma *eint-minus-le-iff*:
fixes $x\ y\ z$ :: *eint*
shows $x - y \leq z \longleftrightarrow$
$(y = -\infty \longrightarrow z = \infty) \wedge$
$(y = \infty \longrightarrow x = \infty \longrightarrow z = \infty) \wedge$
$(|y| \neq \infty \longrightarrow x \leq z + y)$
by (*cases rule*: *eint3-cases*[*of x y z*]) *auto*

lemma *eint-minus-le*:
fixes $x\ y\ z$ :: *eint*
shows $|y| \neq \infty \Longrightarrow x - y \leq z \longleftrightarrow x \leq z + y$
by (*auto simp*: *eint-minus-le-iff*)

lemma *eint-minus-eq-minus-iff*:
fixes $a\ b\ c$ :: *eint*
shows $a - b = a - c \longleftrightarrow$
$b = c \vee a = \infty \vee (a = -\infty \wedge b \neq -\infty \wedge c \neq -\infty)$
by (*cases rule*: *eint3-cases*[*of a b c*]) *auto*

lemma *eint-add-le-add-iff*:
fixes $a\ b\ c$ :: *eint*
shows $c + a \leq c + b \longleftrightarrow$
$a \leq b \vee c = \infty \vee (c = -\infty \wedge a \neq \infty \wedge b \neq \infty)$

by (*cases rule*: *eint3-cases*[*of a b c*]) (*simp-all add*: *field-simps*)

lemma *eint-add-le-add-iff2*:
fixes $a\ b\ c :: eint$
shows $a + c \leq b + c \longleftrightarrow a \leq b \vee c = \infty \vee (c = -\infty \wedge a \neq \infty \wedge b \neq \infty)$
by(*cases rule*: *eint3-cases*[*of a b c*])(*simp-all add*: *field-simps*)

lemma *eint-mult-le-mult-iff*:
fixes $a\ b\ c :: eint$
shows $|c| \neq \infty \Longrightarrow c * a \leq c * b \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
by (*cases rule*: *eint3-cases*[*of a b c*]) (*simp-all add*: *mult-le-cancel-left*)

lemma *eint-minus-mono*:
fixes $A\ B\ C\ D :: eint$ **assumes** $A \leq B\ D \leq C$
shows $A - C \leq B - D$
using *assms*
by (*cases rule*: *eint3-cases*[*case-product eint-cases*, *of A B C D*]) *simp-all*

lemma *eint-mono-minus-cancel*:
fixes $a\ b\ c :: eint$
shows $c - a \leq c - b \Longrightarrow 0 \leq c \Longrightarrow c < \infty \Longrightarrow b \leq a$
by (*cases a b c rule*: *eint3-cases*) *auto*

lemma *int-of-eint-minus*:
fixes $a\ b :: eint$
shows *int-of-eint* $(a - b) =$ (*if* $|a| = \infty \vee |b| = \infty$ *then 0 else int-of-eint a* $-$ *int-of-eint b*)
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *int-of-eint-minus'*: $|x| = \infty \longleftrightarrow |y| = \infty \Longrightarrow$ *int-of-eint x* $-$ *int-of-eint y* $=$ *int-of-eint* $(x - y ::$ *eint*)
by(*subst int-of-eint-minus*) *auto*

lemma *eint-diff-positive*:
fixes $a\ b ::$ *eint* **shows** $a \leq b \Longrightarrow 0 \leq b - a$
by (*cases rule*: *eint2-cases*[*of a b*]) *auto*

lemma *eint-between*:
fixes $x\ e ::$ *eint*
assumes $|x| \neq \infty$
and $0 < e$
shows $x - e < x$
and $x < x + e$
using *assms* **by** (*cases x*, *cases e*, *auto*)+

lemma *eint-minus-eq-PInfty-iff*:
fixes $x\ y ::$ *eint*
shows $x - y = \infty \longleftrightarrow y = -\infty \vee x = \infty$
by (*cases x y rule*: *eint2-cases*) *simp-all*

lemma *eint-diff-add-eq-diff-diff-swap*:

fixes $x\ y\ z :: eint$
shows $|y| \neq \infty \Longrightarrow x - (y + z) = x - y - z$
by(*cases x y z rule*: *eint3-cases*) *simp-all*

lemma *eint-diff-add-assoc2*:
fixes $x\ y\ z :: eint$
shows $x + y - z = x - z + y$
by(*cases x y z rule*: *eint3-cases*) *simp-all*

lemma *eint-add-uminus-conv-diff*: **fixes** $x\ y\ z :: eint$ **shows** $- x + y = y - x$
by(*cases x y rule*: *eint2-cases*) *simp-all*

lemma *eint-minus-diff-eq*:
fixes $x\ y :: eint$
shows $[\![x = \infty \longrightarrow y \neq \infty; x = -\infty \longrightarrow y \neq -\infty]\!] \Longrightarrow - (x - y) = y - x$
by(*cases x y rule*: *eint2-cases*) *simp-all*

lemma *ediff-le-self* [*simp*]: $x - y \leq (x :: enat)$
by(*cases x y rule*: *enat.exhaust*[*case-product enat.exhaust*]) *simp-all*

lemma *eint-abs-diff*:
fixes $a\ b$::*eint*
shows $abs(a-b) \leq abs\ a + abs\ b$
by (*cases rule*: *eint2-cases*[*of a b*]) (*auto*)

1.4.1 Distributive lattice

instantiation *eint* :: *distrib-lattice*
begin

definition [*simp*]: $sup\ x\ y = (max\ x\ y :: eint)$
definition [*simp*]: $inf\ x\ y = (min\ x\ y :: eint)$
instance by *standard*
(*auto simp add*: *inf-int-def sup-int-def max-min-distrib2*)

end

lemma *min-PInf* [*simp*]: $min\ (\infty$::*eint*$)\ x = x$
by *simp*

lemma *min-PInf2* [*simp*]: $min\ x\ (\infty$::*eint*$) = x$
by *simp*

lemma *max-PInf* [*simp*]: $max\ (\infty$::*eint*$)\ x = \infty$
by *simp*

lemma *max-PInf2* [*simp*]: *max x* (∞::*eint*) = ∞
by *simp*

lemma *min-MInf* [*simp*]: *min* (−∞::*eint*) *x* = −∞
by *simp*

lemma *min-MInf2* [*simp*]: *min x* (−∞::*eint*) = −∞
by *simp*

lemma *max-MInf* [*simp*]: *max* (−∞::*eint*) *x* = *x*
by *simp*

lemma *max-MInf2* [*simp*]: *max x* (−∞::*eint*) = *x*
by *simp*

instantiation *eint* :: {*order-bot*, *order-top*}
begin

definition *bot-eint* :: *eint* **where** *bot-eint* = −∞
definition *top-eint* :: *eint* **where** *top-eint* = ∞

instance
by *standard* (*simp-all add*: *bot-eint-def top-eint-def*)

end

lemma *finite-eint-bounded-below-above*:
assumes *le-fin*: ⋀*y*. *y* ∈ *A* ⟹ *eint m* ≤ *y* ∧ *y* ≤ *eint n*
shows *finite A*
proof (*rule finite-subset*)
show *finite* (*eint* ‘ {*m*..*n*}) **by** *blast*
have *A* ⊆ {*eint m*..*eint n*} **using** *le-fin* **by** *fastforce*
also have ... ⊆ *eint* ‘ {*m*..*n*}
apply (*rule subsetI*)
subgoal for *x* **by** (*cases x*) *auto*
done
finally show *A* ⊆ *eint* ‘ {*m*..*n*} **.**
qed

lemma *infinite-eint-bounded-above*:
infinite (*eint* ‘ {..*n*})
by (*meson finite-imageD infinite-Iic inj-eint*)

lemma *infinite-eint-bounded-below*:
infinite (*eint* ‘ {*n*..})

by (*simp add*: *finite-image-iff infinite-Ici*)

lemma *infinite-eint-unbounded-below-or-above*:
assumes *infinite A*
shows $(\exists\ y \in A.\ y > eint\ n) \vee (\exists\ y \in A.\ eint\ m > y)$
using *assms finite-eint-bounded-below-above*[*of A m n*]
by *fastforce*

1.4.2 Extended int intervals

lemma *int-greaterThanLessThan-infinity-eq*:
int-of-eint ‘ $\{N::eint<..<\infty\}$ =
(*if* $N = \infty$ *then* $\{\}$ *else if* $N = -\infty$ *then UNIV else* {*int-of-eint N*<..})
by (*force simp*: *int-less-eint-iff intro*!: *image-eqI*[**where** *x*=*eint* -] *elim*!: *less-eint.elims*)

lemma *int-greaterThanLessThan-minus-infinity-eq*:
int-of-eint ‘ $\{-\infty<..<N::eint\}$ =
(*if* $N = \infty$ *then UNIV else if* $N = -\infty$ *then* $\{\}$ *else* {..<*int-of-eint N*})
proof −
have *int-of-eint* ‘ $\{-\infty<..<N::eint\}$ = *uminus* ‘ *int-of-eint* ‘ $\{-N<..<\infty\}$
by (*auto simp*: *eint-uminus-less-reorder intro*!: *image-eqI*[**where** *x*=−*x* **for** *x*])
also note *int-greaterThanLessThan-infinity-eq*
finally show *?thesis*
using *eint-uminus-eq-reorder int-greaterThanLessThan-infinity-eq* **by** *auto*
qed

lemma *int-greaterThanLessThan-inter*:
int-of-eint ‘ $\{N<..<M::eint\}$ = *int-of-eint* ‘ $\{-\infty<..<M\} \cap$ *int-of-eint* ‘ $\{N<..<\infty\}$
by (*force elim*!: *less-eint.elims*)

lemma *int-atLeastGreaterThan-eq*: *int-of-eint* ‘ $\{N<..<M::eint\}$ =
(*if* $N = \infty$ *then* $\{\}$ *else*
if $N = -\infty$ *then*
(*if* $M = \infty$ *then UNIV*
else if $M = -\infty$ *then* $\{\}$
else {..< *int-of-eint M*})
else if $M = -\infty$ *then* $\{\}$
else if $M = \infty$ *then* {*int-of-eint N*<..}
else {*int-of-eint N* <..< *int-of-eint M*})
proof (*cases* $M = -\infty \vee M = \infty \vee N = -\infty \vee N = \infty$)
case *True*
then show *?thesis*
by (*auto simp*: *int-greaterThanLessThan-minus-infinity-eq int-greaterThanLessThan-infinity-eq*)
next
case *False*
then obtain *p q* **where** $M = eint\ p\ N = eint\ q$
by (*metis MInfty-eq-minfinity eint.distinct*(*3*) *uminus-eint.elims*)
moreover have $\bigwedge x.\ [\![q < x; x < p]\!] \Longrightarrow x \in$ *int-of-eint* ‘ {*eint q*<..<*eint p*}
by (*metis greaterThanLessThan-iff imageI less-eint.simps*(*1*) *int-of-eint.simps*(*1*))
ultimately show *?thesis*

by (*auto elim*!: *less-eint.elims*)
qed

lemma *int-image-eint-ivl*:
fixes *a b*::*eint*
shows
int-of-eint ' {*a*<..<*b*} =
(*if a* < *b then* (*if a* = − ∞ *then if b* = ∞ *then UNIV else* {..<*int-of-eint b*}
else if b = ∞ *then* {*int-of-eint a*<..} *else* {*int-of-eint a* <..< *int-of-eint b*}) *else* {})
by (*cases a*; *cases b*; *simp add*: *int-atLeastGreaterThan-eq not-less*)

lemma fixes *a b c*::*eint*
shows *not-inftyI*: *a* < *b* ⟹ *b* < *c* ⟹ *abs b* ≠ ∞
by *force*

lemma
interval-neqs:
fixes *r s t*::*int*
shows {*r*<..<*s*} ≠ {*t*<..}
and {*r*<..<*s*} ≠ {..<*t*}
and {*r*<..<*ra*} ≠ *UNIV*
and {*r*<..} ≠ {..<*s*}
and {*r*<..} ≠ *UNIV*
and {..<*r*} ≠ *UNIV*
and {} ≠ {*r*<..}
and {} ≠ {..<*r*}
subgoal
by (*metis dual-order.strict-trans greaterThanLessThan-iff greaterThan-iff gt-ex not-le order-refl*)
subgoal
by (*metis finite-greaterThanLessThan-int infinite-Iio*)
subgoal by *force*
subgoal
by (*metis greaterThan-iff lessThan-iff lt-ex order-less-le order-less-trans*)
subgoal by *force*
subgoal by *force*
subgoal using *greaterThan-non-empty* **by** *blast*
subgoal using *lessThan-non-empty* **by** *blast*
done

lemma *int-greaterThanLessThan-empty-iff*:
fixes *a b* :: *int*
shows ({*a*<..<*b*} = {}) ⟷ *a*+*1* ≥ *b*
using *atLeastPlusOneLessThan-greaterThanLessThan-int*
by *fastforce*

lemma *greaterThanLessThan-eq-iff*:
fixes *r s t u*::*int*
shows ({*r*<..<*s*} = {*t*<..<*u*}) = (*r*+*1* ≥ *s* ∧ *u* ≤ *t*+*1* ∨ *r* = *t* ∧ *s* = *u*)

using *int-greaterThanLessThan-empty-iff*[*of r s*] *int-greaterThanLessThan-empty-iff*[*of t u*]
apply *auto*
apply (*metis greaterThanLessThan-iff not-less-iff-gr-or-eq order-less-trans*)
apply (*metis greaterThanLessThan-iff not-less-iff-gr-or-eq order-less-trans*)
apply (*metis greaterThanLessThan-iff not-less-iff-gr-or-eq order-less-trans*)
by (*metis dual-order.strict-trans greaterThanLessThan-iff not-less-iff-gr-or-eq*)

lemma *int-of-eint-image-greaterThanLessThan-iff*:
int-of-eint ‘ {*a* <..< *b*} = *int-of-eint* ‘ {*c* <..< *d*} ⟷ (*a*+*1* ≥ *b* ∧ *c*+*1* ≥ *d* ∨ *a* = *c* ∧ *b* = *d*)
unfolding *int-atLeastGreaterThan-eq*
apply (*cases a*; *cases b*; *cases c*; *cases d*;
simp add: *greaterThanLessThan-eq-iff interval-neqs interval-neqs*[*symmetric*])
using *int-greaterThanLessThan-empty-iff* **by** *blast*+

lemma *uminus-image-int-of-eint-image-greaterThanLessThan*:
uminus ‘ *int-of-eint* ‘ {*l* <..< *u*} = *int-of-eint* ‘ {−*u* <..< −*l*}
by (*force simp*: *algebra-simps eint-less-uminus-reorder*
eint-uminus-less-reorder intro: *image-eqI*[**where** *x*=−*x* **for** *x*])

lemma *add-image-int-of-eint-image-greaterThanLessThan*:
(+) *c* ‘ *int-of-eint* ‘ {*l* <..< *u*} = *int-of-eint* ‘ {*c* + *l* <..< *c* + *u*}
apply *safe*
subgoal for *x*
using *eint-less-add*[*of c*]
by (*force simp*: *int-of-eint-add add.commute*)
subgoal for - *x*
by (*force simp*: *add.commute int-of-eint-minus eint-minus-less eint-less-minus*
intro: *image-eqI*[**where** *x*=*x* − *c*])
done

lemma *add2-image-int-of-eint-image-greaterThanLessThan*:
(λ*x*. *x* + *c*) ‘ *int-of-eint* ‘ {*l* <..< *u*} = *int-of-eint* ‘ {*l* + *c* <..< *u* + *c*}
using *add-image-int-of-eint-image-greaterThanLessThan*[*of c l u*]
by (*metis add.commute image-cong*)

lemma *minus-image-int-of-eint-image-greaterThanLessThan*:
(−) *c* ‘ *int-of-eint* ‘ {*l* <..< *u*} = *int-of-eint* ‘ {*c* − *u* <..< *c* − *l*}
(**is** *?l* = *?r*)
proof −
have *?l* = (+) *c* ‘ *uminus* ‘ *int-of-eint* ‘ {*l* <..< *u*} **by** *auto*
also note *uminus-image-int-of-eint-image-greaterThanLessThan*
also note *add-image-int-of-eint-image-greaterThanLessThan*
finally show *?thesis* **by** (*simp add*: *minus-eint-def*)
qed

lemma *int-eint-bound-lemma-up*:
assumes *s* ∈ *int-of-eint* ‘ {*a*<..<*b*}

assumes $t \notin$ *int-of-eint* ` $\{a<..<b\}$
assumes $s \leq t$
shows $b \neq \infty$
proof (*cases b*)
case *PInf*
then show *?thesis*
using *assms*
apply *clarsimp*
by (*metis UNIV-I assms*(*1*) *eint-less-PInfty greaterThan-iff less-eq-eint-def less-le-trans int-image-eint-ivl*)
qed *auto*

lemma *int-eint-bound-lemma-down*:
assumes *s*: $s \in$ *int-of-eint* ` $\{a<..<b\}$
and *t*: $t \notin$ *int-of-eint* ` $\{a<..<b\}$
and $t \leq s$
shows $a \neq -\infty$
proof (*cases b*)
case (*int r*)
then show *?thesis*
using *assms int-greaterThanLessThan-minus-infinity-eq* **by** *force*
next
case *PInf*
then show *?thesis*
using *t int-greaterThanLessThan-infinity-eq* **by** *auto*
next
case *MInf*
then show *?thesis*
using *s* **by** *auto*
qed

lemma *eint-less-le-alt*:
$eint\ i < j \Longrightarrow eint\ (i+1) \leq j$
using *eint-less-eq*(*3*)[*of i+1*]
by (*metis eint-less-eint-Ex eint-less-le less-iff-succ-less-eq linorder-not-less order.asym*)

lemma *exists-eint-int*:
assumes $i \neq -\infty$
$i \neq \infty$
shows $\exists n.\ i = eint\ n$
using *assms*
by (*metis PInfty-eq-infinity eint.exhaust uminus-eint.simps*(*2*))

1.4.3 Relation to *enat*

definition *eint-of-enat* $n = (case\ n\ of\ enat\ n \Rightarrow eint\ (int\ n) \mid \infty \Rightarrow \infty)$

declare [[*coercion eint-of-enat* :: *enat* $\Rightarrow$ *eint*]]
declare [[*coercion* ($\lambda n.$ *eint* (*int n*)) :: *nat* $\Rightarrow$ *eint*]]

lemma *eint-of-enat-simps*[*simp*]:
eint-of-enat (*enat n*) = *eint n*
eint-of-enat ∞ = ∞
by (*simp-all add*: *eint-of-enat-def*)

lemma *eint-of-enat-le-iff*[*simp*]:
eint-of-enat m ≤ *eint-of-enat n* ⟷ *m* ≤ *n*
by (*cases m n rule*: *enat2-cases*) *auto*

lemma *eint-of-enat-less-iff*[*simp*]:
eint-of-enat m < *eint-of-enat n* ⟷ *m* < *n*
by (*cases m n rule*: *enat2-cases*) *auto*

lemma *numeral-le-eint-of-enat-iff*[*simp*]:
numeral m ≤ *eint-of-enat n* ⟷ *numeral m* ≤ *n*
by (*cases n*) *auto*

lemma *numeral-less-eint-of-enat-iff*[*simp*]:
numeral m < *eint-of-enat n* ⟷ *numeral m* < *n*
by (*cases n*) *auto*

lemma *eint-of-enat-ge-zero-cancel-iff*[*simp*]:
0 ≤ *eint-of-enat n* ⟷ *0* ≤ *n*
by (*cases n*) (*auto simp flip*: *enat-0*)

lemma *eint-of-enat-gt-zero-cancel-iff*[*simp*]:
0 < *eint-of-enat n* ⟷ *0* < *n*
by (*cases n*) (*auto simp flip*: *enat-0*)

lemma *eint-of-enat-zero*[*simp*]:
eint-of-enat 0 = *0*
by (*auto simp flip*: *enat-0*)

lemma *eint-of-enat-inf*[*simp*]:
eint-of-enat n = ∞ ⟷ *n* = ∞
by (*cases n*) *auto*

lemma *eint-of-enat-add*:
eint-of-enat (*m* + *n*) = *eint-of-enat m* + *eint-of-enat n*
by (*cases m n rule*: *enat2-cases*) *auto*

lemma *eint-of-enat-sub*:
assumes *n* ≤ *m*
shows *eint-of-enat* (*m* − *n*) = *eint-of-enat m* − *eint-of-enat n*
using *assms* **by** (*cases m n rule*: *enat2-cases*) *auto*

lemma *eint-of-enat-mult*:

eint-of-enat $(m * n)$ = *eint-of-enat* m $*$ *eint-of-enat* n
by (*cases* m n *rule*: *enat2-cases*) *auto*

lemmas *eint-of-enat-pushin* = *eint-of-enat-add eint-of-enat-sub eint-of-enat-mult*
lemmas *eint-of-enat-pushout* = *eint-of-enat-pushin*[*symmetric*]

lemma *eint-of-enat-nonneg*: *eint-of-enat* $n \geq 0$
by(*cases* n) *simp-all*

end

Chapter 2

Update

theory *Update*
imports *NELList-Extras HOL−Library.FSet*

begin

This theory formalises the upd function which is used to model quantification in the deep embedding of temporal logics.

2.1 Intervals

type-synonym *interval* = *nat fset nellist*

2.2 Semantics Primitive Operators

2.2.1 Definitions

definition *similar* :: *interval* ⇒ *nat* ⇒ *interval* ⇒ *bool* ((- ∼- -))
where *similar* $\sigma 0\ n\ \sigma 1 = (nlength\ \sigma 0 = nlength\ \sigma 1 \wedge$
$(\forall\ k\ p\ .\ enat\ k \leq nlength\ \sigma 0 \longrightarrow$
$(p\ |\in|\ (nnth\ \sigma 0\ k) \wedge p \neq n) \longleftrightarrow (p\ |\in|\ (nnth\ \sigma 1\ k) \wedge p \neq n)\ )\ )$

definition *similars* :: *interval* ⇒ *nat fset* ⇒ *interval* ⇒ *bool* ((- ≈- -))
where *similars* $\sigma 0\ A\ \sigma 1 = (nlength\ \sigma 0 = nlength\ \sigma 1 \wedge$
$(\forall\ k\ p.\ k \leq nlength\ \sigma 0 \longrightarrow\ p\ |\notin|\ A \longrightarrow (p\ |\in|\ (nnth\ \sigma 0\ k) \longleftrightarrow p\ |\in|\ (nnth\ \sigma 1\ k))))$

definition *upd-fun*
where *upd-fun n* = (λ (*x*,*y*). (*if y then finsert n x else x* |−| {|*n*|}))

definition *upd* :: *interval* ⇒ *nat* ⇒ (*bool nellist*) ⇒ *interval*
where
upd σ *n l* = *nmap* (λ (*x*,*y*). (*if y then finsert n x else x* |−| {|*n*|})) (*nzip* σ *l*)

definition *sfxfilt* :: *interval* ⇒ (*interval* ⇒ *bool*) ⇒ *interval nellist*
where *sfxfilt s P* = (*nfilter P* (*ndropns s*))

definition *pifilt* :: *interval* ⇒ (*interval* ⇒ *bool*) ⇒ *interval*

where *pifilt s f* = (*nmap* (λ*s*. *nnth s 0*) (*sfxfilt s f*))

definition *piwfilt* :: *interval* ⇒ (*interval* ⇒ *bool*) ⇒ *interval*
where *piwfilt s f* = *nfilter* (λ *c* . *f* (*NNil c*)) *s*

primcorec *pfilt* :: *interval* ⇒ *nat nellist* ⇒ *interval*
where
pfilt σ ls = (*case ls of* (*NNil n*) ⇒ (*NNil* (*nnth σ n*)) |
(*NCons n ls1*) ⇒ (*NCons* (*nnth σ n*) (*pfilt σ ls1*)))

simps-of-case *pfilt-code* [*code*, *simp*, *nitpick-simp*]: *pfilt.code*

definition *inv-X-fun*
where *inv-X-fun i σ ll v P* = (*if i* = (*nnth* (*nkfilter* (*P*∘*fst*) *0* ((*nzip* (*ndropns σ*) (*ndropns ll*)))) *i*) *then v else* (*nnth ll i*))

definition *inv-pi-X*
where *inv-pi-X σ ul P* = (*SOME ul1*. *nlength ul1* = *nlength σ* ∧ *ul* = *nmap* (λ*s*. (*nnth s 0*)) (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns σ*) (*ndropns ul1*)))))

primcorec *cnt-in-nellist*
where *cnt-in-nellist nell P n* =
(*case nell of* (*NNil b*) ⇒ (*if P b then* (*NNil* (*b*, *Suc n*)) *else* (*NNil* (*b*, *n*))) |
(*NCons x nell*′) ⇒ (*if P x then* (*NCons* (*x*, *Suc n*) (*cnt-in-nellist nell*′ *P* (*Suc n*)))
else (*NCons* (*x*, *n*) (*cnt-in-nellist nell*′ *P n*))))

simps-of-case *cnt-in-nellist-code* [*code*,*simp*, *nitpick-simp*]: *cnt-in-nellist.code*

definition *inv-pi-X-concrete*
where *inv-pi-X-concrete σ ul P* =
nmap (λ(*x*, *k*). *if P x then nnth ul* (*k* − *1*) *else True*) (*cnt-in-nellist* (*ndropns σ*) *P 0*)

2.2.2 Lemmas

general

lemma *exists-nellist-nnth-fun*:
∃ *l*. (∀ *i*. *i* ≤ *nlength l* ⟶ (*nnth l i*) = (*F-X i*))
by (*metis enat-0-iff*(*2*) *ile0-eq nlength-NNil nnth-NNil*)

lemma *nfilter-ntaken-a*:
assumes *i* ≤ *nlength nell*
P (*nnth nell i*)
∀ *j*. *j* < *i* ⟶ ¬ *P* (*nnth nell j*)
shows *nlength* (*nfilter P* (*ntaken i nell*)) = *0*
proof −
have *1*: (∃ *x*∈*nset* (*ntaken i nell*). *P x*)
using *assms* **by** (*metis nfinite-ntaken nset-nlast ntaken-nlast*)
have *2*: ((∃ *x*∈*nset* (*ntaken i nell*). *P x*) ∧ *nlength* (*nfilter P* (*ntaken i nell*)) = *0*) =

$(\exists\, k.\ enat\ k \leq nlength\ (ntaken\ i\ nell) \wedge$
$P\ (nnth\ (ntaken\ i\ nell)\ k) \wedge (\forall\, j.\ enat\ j \leq nlength\ (ntaken\ i\ nell) \longrightarrow j < k \vee k < j \longrightarrow \neg\ P\ (nnth\ (ntaken\ i\ nell)\ j)))$
using *nfilter-nlength-zero-conv-1*[*of ntaken i nell P*] **by** *blast*
show *?thesis* **using** *assms*
by (*metis 2 antisym-conv2 dual-order.refl enat-ord-simps*(*1*) *min.orderE ntaken-nlength ntaken-nnth*)
qed

lemma *nfilter-ntaken-b*:
assumes $i \leq nlength\ nell$
$\exists\ j.\ j \leq i \longrightarrow\ P\ (nnth\ nell\ j)$
shows $nlength\ (nfilter\ P\ (ntaken\ i\ nell)) \geq 0$
by *auto*

lemma *sfxfilter-help*:
$(\exists\ ys \in nset\ (ndropns\ xs)\ .\ f\ ys) = (\exists\ i \leq nlength\ xs.\ f\ (ndropn\ i\ xs))$
using *in-nset-ndropns* **by** *auto*

lemma *pifiltinit-help*:
$(\exists\ y \in nset\ (xs).\ w\ (NNil\ y)\ ) = (\exists\ i \leq nlength\ xs.\ w\ (NNil\ (nnth\ xs\ i)))$
by (*metis in-nset-conv-nnth*)

lemma *sfxfilt-nnth*:
assumes $(\exists\ i \leq nlength\ \sigma.\ P\ (ndropn\ i\ \sigma))$
$i \leq nlength\ (sfxfilt\ \sigma\ P)$
shows $P\ (nnth\ (sfxfilt\ \sigma\ P)\ i)$
using *assms* **by** (*metis in-nset-ndropns nkfilter-nnth-aa sfxfilt-def*)

lemma *pifilt-exists*:
assumes $(\exists\, i \leq nlength\ \sigma.\ P\ (ndropn\ i\ \sigma)\ )$
shows $(\exists\, i \leq nlength(sfxfilt\ \sigma\ P).\ P\ (nnth\ (sfxfilt\ \sigma\ P)\ i)\ \ )$
using *assms* **by** (*metis sfxfilt-nnth zero-enat-def zero-le*)

lemma *sfxfilt-pifilt-nlength*:
shows $nlength\ (pifilt\ \sigma\ P) = nlength\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ P))$
by (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt-nnth*:
assumes $j \leq nlength\ (pifilt\ \sigma\ P)$
shows $nnth\ (pifilt\ \sigma\ P)\ j = nnth\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ P))\ j$
using *assms* **by** (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt*:
shows $(nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ P)) = pifilt\ \sigma\ P$
by (*simp add*: *pifilt-def*)

lemma *piwfilt-eq-pifilt*:
shows $pifilt\ \sigma\ (\lambda s.\ P\ (NNil\ (nnth\ s\ 0))\ ) = piwfilt\ \sigma\ P$
unfolding *piwfilt-def pifilt-def sfxfilt-def*
using *nfilter-nmap*[*of* $(\lambda c.\ P\ (NNil\ c))\ \ (\lambda s.\ nnth\ s\ 0)\ (ndropns\ \sigma)$]

nmap-first-ndropns[*of* σ]
by (*metis* (*mono-tags*, *lifting*) *comp-eq-dest-lhs nfilter-cong*)

lemma *sfxfilt-nlength-bound*:
assumes ($\exists\, i{\leq}$*nlength xs*. P (*ndropn i xs*))
shows *nlength* (*sfxfilt xs P*) $\leq$ *nlength xs*
using *assms*
by (*metis length-nfilter-le ndropns-nlength sfxfilt-def*)

lemma *pifilt-nlength-bound*:
assumes ($\exists\, i{\leq}$*nlength* σ. P (*ndropn i* σ))
shows *nlength* (*pifilt* σ P) $\leq$ *nlength* σ
using *assms* **by** (*simp add*: *pifilt-def sfxfilt-nlength-bound*)

lemma *sfxfilt-nlength-nnth-bound*:
assumes ($\exists\, i{\leq}$*nlength* σ. P (*ndropn i* σ))
$j{\leq}$ *nlength* (*sfxfilt* σ P)
shows *nlength* (*nnth* (*sfxfilt* σ P) j) $\leq$ *nlength* σ
using *assms* **by** (*simp add*: *nfilter-ndropns-nnth-bound sfxfilt-def sfxfilter-help*)

lemma *sfxfilt-pifilt-nnth-ndropn*:
assumes ($\exists$ $i \leq$ *nlength* σ. P (*ndropn i* σ))
$j{\leq}$ *nlength* (*pifilt* σ P)
shows *nnth* (*sfxfilt* σ P) j = *ndropn* (*nnth* (*nkfilter P 0* (*ndropns* σ)) j) σ
proof −
have *0*: $\exists\, x{\in}$*nset* (*ndropns* σ). $P\ x$
by (*simp add*: *assms*(*1*) *sfxfilter-help*)
have *1*: *nnth* (*sfxfilt* σ P) j = *nnth* (*nfilter P* (*ndropns* σ)) j
by (*simp add*: *sfxfilt-def*)
have *2*: *nnth* (*nfilter P* (*ndropns* σ)) j = *nnth* (*ndropns* σ) (*nnth* (*nkfilter P 0* (*ndropns* σ)) j)
using *nkfilter-nfilter*[*of ndropns* σ λ *s*. *P s j 0*] *assms* **unfolding** *pifilt-def sfxfilt-def*
by (*simp add*: *0 nkfilter-nlength*)
have *30*: *enat* (*nnth* (*nkfilter P 0* (*ndropns* σ)) j) $\leq$ *nlength* σ
using *0 nkfilter-upperbound*[*of* (*ndropns* σ) *P j 0*] *assms* **unfolding** *pifilt-def sfxfilt-def*
by (*metis gen-nlength-def ndropns-nlength nkfilter-nlength nlength-code nlength-nmap*)
have *3*: *nnth* (*ndropns* σ) (*nnth* (*nkfilter P 0* (*ndropns* σ)) j) =
ndropn (*nnth* (*nkfilter P 0* (*ndropns* σ)) j) σ
using *ndropns-nnth*[*of* (*nnth* (*nkfilter P 0* (*ndropns* σ)) j)]
using *30* **by** *blast*
show *?thesis* **by** (*simp add*: *1 2 3*)
qed

lemma *pifilt-nnth-exists*:
assumes ($\exists\, i{\leq}$*nlength* σ. P (*ndropn i* σ))
$i \leq$ *nlength* (*pifilt* σ P)
shows ($\exists$ $k \leq$ *nlength* σ. *nnth* (*pifilt* σ P) i = *nnth* σ k)
using *assms sfxfilt-pifilt-nnth-ndropn*[*of* σ P i]
by (*simp add*: *pifilt-def*)
(*metis enat-the-enat infinity-ileE linorder-le-cases ndropn-all ndropn-nfirst*)

lemma *pifilt-nnth*:
assumes ($\exists$ *i*$\leq$*nlength* σ. *P* (*ndropn i* σ))
i $\leq$ *nlength* (*pifilt* σ *P*)
shows *nnth* (*pifilt* σ *P*) *i* =
nnth σ (*nnth* (*nkfilter P 0* (*ndropns* σ)) *i*)
using *assms* **by** (*simp add*: *pifilt-def sfxfilt-pifilt-nnth-ndropn*)

lemma *pfilt-nmap*:
pfilt nell ls = *nmap* (λx. (*nnth nell x*)) *ls*
proof (*coinduction arbitrary*: *nell ls*)
case (*Eq-nellist nell1*)
then show *?case* **by** *simp* (*metis nellist.case-eq-if*)
qed

lemma *pfilt-nlength*:
nlength(*pfilt nell ls*) = *nlength ls*
by (*simp add*: *pfilt-nmap*)

lemma *pfilt-nnth*:
assumes *i* $\leq$ *nlength* (*pfilt nell ls*)
shows (*nnth* (*pfilt nell ls*) *i*) = (*nnth nell* (*nnth ls i*))
using *assms*
by (*simp add*: *pfilt-nmap*)

lemma *pfilt-expand*:
(*nell1* = (*pfilt nell ls*)) =
(*nlength nell1* = *nlength ls* $\wedge$
($\forall$ (*i*::*nat*). *i* $\leq$ *nlength nell1* $\longrightarrow$ (*nnth nell1 i*) = (*nnth nell* (*nnth ls i*))))
by (*metis nellist-eq-nnth-eq pfilt-nlength pfilt-nnth*)

lemma *ndropns-nmap*:
shows *ndropns* σ = (*nmap* (λ (*s*,*i*). *ndropn i* σ) (*nzip* σ (*niterates Suc 0*)))
proof −
have *1*: *nlength* (*ndropns* σ) = *nlength* (*nmap* (λ(*s*,*i*). *ndropn i* σ) (*nzip* σ (*niterates Suc 0*)))
by (*simp add*: *ndropns-nlength*)
have *2*: $\bigwedge$*j*. *j*$\leq$ *nlength* (*ndropns* σ) $\Longrightarrow$
nnth (*ndropns* σ) *j* =
ndropn j σ
by (*simp add*: *ndropns-nlength ndropns-nnth*)
have *2*: $\bigwedge$*j*. *j*$\leq$ *nlength* (*ndropns* σ) $\Longrightarrow$
nnth (*nmap* (λ(*s*,*i*). *ndropn i* σ) (*nzip* σ (*niterates Suc 0*))) *j* =
ndropn j σ
by (*simp add*: *ndropns-nlength nnth-nzip*)
show *?thesis*
by (*metis 1 2 ndropns-nlength ndropns-nnth nellist-eq-nnth-eq*)
qed

lemma *nzip-ndropns-nmap*:
assumes *nlength σ = nlength l*
shows *nmap (λ (s,i). (ndropn i σ, ndropn i l)) (nzip σ (niterates Suc 0)) =*
nzip (ndropns σ) (ndropns l)
proof −
have *1*: *nlength (nzip (ndropns σ) (ndropns l)) =*
nlength (nmap (λ (s,i). (ndropn i σ, ndropn i l)) (nzip σ (niterates Suc 0)))
by (*simp add*: *assms ndropns-nlength*)
have *2*: *⋀j. j ≤ nlength (nzip (ndropns σ) (ndropns l)) ⟹*
nnth (nzip (ndropns σ) (ndropns l)) j =
(ndropn j σ , ndropn j l)
by (*simp add*: *ndropns-nlength ndropns-nnth nnth-nzip*)
have *3*: *⋀j. j ≤ nlength (nzip (ndropns σ) (ndropns l)) ⟹*
nnth (nmap (λ (s,i). (ndropn i σ, ndropn i l)) (nzip σ (niterates Suc 0))) j =
(ndropn j σ , ndropn j l)
proof −
fix *j* ::*nat*
assume *a0*: *j ≤ nlength (nzip (ndropns σ) (ndropns l))*
show *nnth (nmap (λ (s,i). (ndropn i σ, ndropn i l)) (nzip σ (niterates Suc 0))) j =*
(ndropn j σ , ndropn j l)
proof −
have *4*: *nnth (nzip σ (niterates Suc 0)) j = (nnth σ j, nnth (niterates Suc 0) j)*
using *assms a0 1* **apply** *auto*
by (*simp add*: *ndropns-nlength nnth-nzip*)
show *?thesis*
using *nnth-nmap*[*of j (nzip σ (niterates Suc 0)) (λ (s,i). (ndropn i σ, ndropn i l))*]
4 a0 1 **apply** *auto*
by (*simp add*: *ndropns-nlength*)
qed
qed
show *?thesis*
using *1 2 3 nellist-eq-nnth-eq* **by** *force*
qed

lemma *nfilter-ndropns-eq-nmap*:
nfilter (P) (ndropns xs) = nmap (λ (x,y). (ndropn y xs)) (nfilter ((P)∘(λ (x,y). (ndropn y xs))) (nzip xs (niterates Suc 0)))
by (*metis ndropns-nmap nfilter-nmap*)

lemma *nfilter-nzip-ndropns-eq*:
assumes *nlength ls = nlength xs*
shows *nfilter (P∘fst) (nzip (ndropns xs) (ndropns ls)) =*
nmap (λ (x,z). (ndropn z xs, ndropn z ls)) (nfilter ((P∘fst)∘(λ (x,z). (ndropn z xs, ndropn z ls))) (nzip xs (niterates Suc 0)))
proof −
have *1*: *nfilter (P∘fst) (nzip (ndropns xs) (ndropns ls)) =*
nfilter (P∘fst) (nmap (λ (x,z). (ndropn z xs, ndropn z ls)) (nzip xs (niterates Suc 0)))
using *nzip-ndropns-nmap* **by** (*metis (mono-tags, lifting*) *assms*)

have *2*: *nfilter* (*P*∘*fst*) (*nmap* (λ (*x*,*z*). (*ndropn z xs*, *ndropn z ls*)) (*nzip xs* (*niterates Suc 0*))) =
nmap (λ (*x*,*z*). (*ndropn z xs*, *ndropn z ls*)) (*nfilter* ((*P*∘*fst*)∘ (λ (*x*,*z*). (*ndropn z xs*, *ndropn z ls*)))
(*nzip xs* (*niterates Suc 0*)))
by (*simp add*: *nfilter-nmap*)
show *?thesis*
by (*simp add*: *1 2*)
qed

cnt_in_nellist_simps

lemma *cnt-in-nellist-simps* [*simp*]:
¬ *is-NNil nell* ⟹ *nhd* (*cnt-in-nellist nell P n*) =
(*if P* (*nhd nell*) *then* ((*nhd nell*, *Suc n*)) *else* ((*nhd nell*, *n*)))
ntl(*cnt-in-nellist nell P n*) =
(*case nell of* (*NNil b*) ⇒ (*if P b then* (*NNil* (*b*, *Suc n*)) *else* (*NNil* (*b*, *n*))) |
(*NCons x nell*′) ⇒
(*if P x then* (*cnt-in-nellist nell*′ *P* (*Suc n*)) *else* (*cnt-in-nellist nell*′ *P n*)))
by (*auto simp add*: *cnt-in-nellist.code nellist.case-eq-if nat.case-eq-if*)

lemma *cnt-in-nellist-nnth*:
assumes *i* ≤ *nlength nell*
shows *nnth* (*cnt-in-nellist nell P n*) *i* =
(*if* (∀ *j*. *j* ≤ *i* ⟶ ¬ *P* (*nnth nell j*))
then (*nnth nell i*, *n*)
else (*nnth nell i*, *n*+*the-enat* (*nlength* (*nfilter P* (*ntaken i nell*))) + *1*))
using *assms*
proof (*induction i arbitrary*: *nell n*)
case *0*
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **using** *0* **by** (*auto simp add*: *nnth-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis* **using** *NCons* **by** *auto*
qed
next
case (*Suc i*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **using** *Suc* **by** (*auto simp add*: *nnth-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis*
proof (*cases P x21*)
case *True*
then show *?thesis*

```
      proof (cases ∀ j≤ i. ¬ P (nnth nell j) )
      case True
      then show ?thesis using Suc NCons ‹P x21› by auto
      next
      case False
      then show ?thesis
        proof −
        have p0: ∃ j. j ≤ i ∧ P (nnth nell j)
          using False by auto
        have p1: nnth (cnt-in-nellist (NCons x21 x22) P n) (Suc i) =
                 nnth (cnt-in-nellist x22 P (Suc n)) i
         using True  by simp
         have p2: nnth (NCons x21 x22) (Suc i)  = (nnth x22 i)
          by simp
         have p3: (nfilter P (ntaken (Suc i) (NCons x21 x22))) =
                  nfilter P (NCons x21 (ntaken i x22))
              by simp
         have p30: (∃ x ∈ nset (ntaken i x22). P x) ⟹ ¬(∀ j≤Suc i. ¬ P (nnth nell j))
              using le-Suc-eq p0 by auto
         have p31: (∃ x ∈ nset (ntaken i x22). P x) ⟹
                   nnth (cnt-in-nellist x22 P (Suc n)) i = (nnth x22 i, Suc n + the-enat (nlength (nfilter
P (ntaken i x22))) + 1)
            using Suc.IH[of x22 (Suc n) ] Suc.prems
              by (metis NCons Suc-ile-eq enat-ord-simps(1) iless-Suc-eq in-nset-conv-nnth min.orderE
nlength-NCons ntaken-nlength ntaken-nnth)
         have p32: (∃ x ∈ nset (ntaken i x22). P x) ⟹
               Suc n + the-enat (nlength (nfilter P (ntaken i x22))) + 1 =
               n + the-enat (nlength (nfilter P (ntaken (Suc i) nell))) + 1
          using NCons Suc.prems  apply simp
              by (metis True enat-the-enat infinity-ileE  length-nfilter-le ndropn-eq-NNil ndropn-nlast
nfinite-ntaken the-enat-eSuc)
         have p4: (∃ x ∈ nset (ntaken i x22). P x) ⟹ ?thesis
             using Suc.IH[of x22 (Suc n) ] Suc.prems NCons[of ] ‹P x21›
          using p1 p2 p30 p31 p32 by argo
         have p5: ¬(∃ x ∈ nset (ntaken i x22). P x) ⟹ ?thesis
             using Suc NCons ‹P x21› apply auto
                  by (metis eSuc-enat eSuc-ile-mono enat-ord-simps(1) in-nset-conv-nnth min.orderE
ntaken-nlength ntaken-nnth)
         show ?thesis
         using p4 p5 by blast
        qed
       qed
     next
     case False
     then show ?thesis
       proof (cases ∀ j≤ i. ¬ P (nnth nell j) )
       case True
       then show ?thesis
          proof −
           have p0: ∀ j≤Suc i. ¬ P (nnth nell j) ⟹ ?thesis
```

```
            using True NCons Suc ‹¬ P x21› using Suc-ile-eq by auto
          have p1: (∀ j≤Suc i. ¬ P (nnth nell j)) =
                ((∀ j≤ i. ¬ P (nnth nell j)) ∧ (¬ P (nnth x22 i)))
            using le-Suc-eq  using True NCons Suc by simp

          have p2: (∀ j≤ i. ¬ P (nnth nell j)) ∧ P (nnth x22 i) ⟹ ?thesis
            using True NCons Suc.IH[of x22 n]  Suc.prems ‹¬ P x21›  apply auto
             by (metis One-nat-def Suc-ile-eq add-Suc-right iless-Suc-eq in-nset-conv-nnth min.orderE
nat-arith.rule0
           nfilter-NCons nfinite-ntaken nnth-nlast ntaken-nlast ntaken-nlength order.refl the-enat.simps)

          have p3:  (∀ j≤ i. ¬ P (nnth nell j)) ∧ ¬ P (nnth x22 i) ⟹ ?thesis
            using True NCons Suc.IH[of x22 n] Suc.prems ‹¬ P x21›  apply auto
            using p0 apply auto[1]
            using p1 by blast
          show ?thesis
          using True p2 p3 by blast
        qed
      next
      case False
      then show ?thesis
        proof −
        have np0: ∃ j. j ≤ i ∧ P (nnth nell j)
         using False by auto
        have np1: nnth (cnt-in-nellist (NCons x21 x22) P n) (Suc i) =
              nnth (cnt-in-nellist x22 P n) i
        using False ‹¬ P x21› by simp
        have np2: nnth (NCons x21 x22) (Suc i)  = (nnth x22 i)
          by simp
        have np3: (nfilter P (ntaken (Suc i) (NCons x21 x22))) =
              nfilter P (NCons x21 (ntaken i x22))
            by simp
        have np4: (∃  x ∈ nset (ntaken i x22). P x) ⟹ ?thesis
          using Suc NCons ‹¬ P x21› False apply auto
          by (metis add.commute eSuc-enat eSuc-ile-mono enat-ord-simps(1) in-nset-conv-nnth
              min.orderE ntaken-nlength ntaken-nnth plus-1-eq-Suc)
        have np5: ¬(∃  x ∈ nset (ntaken i x22). P x) ⟹ ?thesis
          proof −
          assume anp: ¬(∃  x ∈ nset (ntaken i x22). P x)
          show ?thesis
          proof −
           have np6: ∀  j. j ≤ i⟶ ¬ P (nnth x22 j)
            using anp
            using NCons Suc.prems Suc-ile-eq ntaken-nset by force
           have np7: ∀  j. j ≤ (Suc i) ⟶ ¬ P (nnth (NCons x21 x22) j)
            using ‹¬ P x21›  np6
              by (metis Suc-le-mono dual-order.strict-iff-order gr0-conv-Suc nnth-0 nnth-Suc-NCons
zero-le)
           show ?thesis
           using False NCons le-Suc-eq np7 by blast
```

```
            qed
           qed
         show ?thesis
         using np4 np5 by blast
        qed
      qed
    qed
  qed
qed

lemma cnt-in-nellist-nlength:
shows  nlength (cnt-in-nellist nell P n) = nlength nell
proof (coinduction arbitrary: nell n rule: enat-coinduct)
case (Eq-enat nell1 n1)
then show ?case
proof −
  have 1: (nlength (cnt-in-nellist nell1 P n1) = 0) = (nlength nell1 = 0)
    by (metis cnt-in-nellist.disc-iff(2) is-NNil-def nlength-NNil ntaken-0 ntaken-all zero-le)
  have 2: (nlength (cnt-in-nellist nell1 P n1) ≠ 0 ⟶
     nlength nell1 ≠ 0 ⟶
    (∃ nell n. epred (nlength (cnt-in-nellist nell1 P n1)) = nlength (cnt-in-nellist nell P n) ∧ epred (nlength
nell1) = nlength nell) ∨
     epred (nlength (cnt-in-nellist nell1 P n1)) = epred (nlength nell1))
      using nellist-split-2-first[of nell1] apply simp
       by (metis cnt-in-nellist-code(2) co.enat.sel(2) nlength-NCons)
  show ?thesis using 1 2 by auto
  qed
qed

lemma cnt-in-nellist-nmap:
  (cnt-in-nellist (nmap f nell) P n) =  (nmap (λ (x, y). (f x , y)) (cnt-in-nellist nell (P∘f) n))
proof −
 have 1: nlength (cnt-in-nellist (nmap f nell) P n) = nlength (nmap (λ (x, y). (f x , y)) (cnt-in-nellist nell
(P∘f) n))
   by (simp add: cnt-in-nellist-nlength)
 have 2: ⋀j. j≤ nlength (cnt-in-nellist (nmap f nell) P n) ⟹
          nnth (cnt-in-nellist (nmap f nell) P n) j =
          nnth (nmap (λ (x, y). (f x , y)) (cnt-in-nellist nell (P∘f) n)) j
   proof −
    fix j ::nat
    assume a: j≤ nlength (cnt-in-nellist (nmap f nell) P n)
    show nnth (cnt-in-nellist (nmap f nell) P n) j =
          nnth (nmap (λ (x, y). (f x , y)) (cnt-in-nellist nell (P∘f) n)) j
    proof −
    have 20: enat j ≤ nlength (cnt-in-nellist nell (P ∘ f) n)
       using a          by (simp add: 1)
    have 21: nnth (cnt-in-nellist (nmap f nell) P n) j =
            (if ∀ ja≤j. ¬ P (nnth (nmap f nell) ja) then (nnth (nmap f nell) j, n)
             else (nnth (nmap f nell) j, n + the-enat (nlength (nfilter P (ntaken j (nmap f nell)))) + 1))
     using  a  by (simp add: cnt-in-nellist-nlength cnt-in-nellist-nnth)
```

```
  have 22: (∀ ja≤j. ¬ P (nnth (nmap f nell) ja)) = (∀ ja≤j. ¬ P (f (nnth nell ja)))
    using nnth-nmap[of - nell f] using a
    by (simp add: cnt-in-nellist-nlength order-subst2)
  have 23: (nnth (nmap f nell) j, n) =  (f (nnth nell j), n)
     using nnth-nmap[of - nell f] using a
     by (simp add: cnt-in-nellist-nlength)
  have 24: (nnth (nmap f nell) j, n + the-enat (nlength (nfilter P (ntaken j (nmap f nell)))) + 1) =
          (f (nnth nell j), n + the-enat (nlength (nfilter P ( (nmap f (ntaken j nell))))) + 1)
   using nnth-nmap[of - nell f] using a  using 23 by fastforce
  have 25: nnth (nmap (λ(x, y). (f x, y)) (cnt-in-nellist nell (P ∘ f) n)) j =
          (λ(x, y). (f x, y)) (nnth (cnt-in-nellist nell (P ∘ f) n) j)
   using nnth-nmap[of j (cnt-in-nellist nell (P ∘ f) n) (λ(x, y). (f x, y))] a
    by (simp add: 20)
  have 26: (nnth (cnt-in-nellist nell (P ∘ f) n) j) =
          (if ∀ ja≤j. ¬ (P ∘ f) (nnth nell ja) then (nnth nell j, n)
           else (nnth nell j, n + the-enat (nlength (nfilter (P ∘ f) (ntaken j nell))) + 1))
    using  cnt-in-nellist-nnth[of j nell (P ∘ f) n] a  by (simp add: cnt-in-nellist-nlength)
  have 27: (λ(x, y). (f x, y)) (nnth (cnt-in-nellist nell (P ∘ f) n) j) =
          (if ∀ ja≤j. ¬ (P ∘ f) (nnth nell ja) then (f (nnth nell j), n)
           else (f (nnth nell j), n + the-enat (nlength (nfilter (P ∘ f) (ntaken j nell))) + 1))
    by (simp add: 26)
  have 28: (∀ ja≤j. ¬ (P ∘ f) (nnth nell ja)) =  (∀ ja≤j. ¬ P (f (nnth nell ja)))
     by simp
  have 29: (n + the-enat (nlength (nfilter P ( (nmap f (ntaken j nell))))) + 1) =
          (n + the-enat (nlength (nfilter (P ∘ f) (ntaken j nell))) + 1)
       by auto
  show ?thesis
  using 21 22 23 24 25 27 28 29
  by presburger
 qed
qed
show ?thesis
by (simp add: 1 2 nellist-eq-nnth-eq)
qed

lemma cnt-in-nellist-nfilter:
assumes ∃ x ∈ nset nell. P x
      ll = nmap (λ (x,k). if P x then (nnth l (k − n − 1)) else True ) (cnt-in-nellist nell P n)
     nlength l = nlength (nmap snd (nfilter (P∘fst) (nzip nell ll)))
shows  l = nmap snd (nfilter (P∘fst) (nzip nell ll))
proof −
 have 1: nlength ll = nlength nell
   using assms    by (simp add: cnt-in-nellist-nlength)
 have 2: ⋀j . j ≤ nlength ll ⟹
          nnth ll j =
          (if P (nnth nell j) then
          nnth l ( the-enat (nlength (nfilter P (ntaken j nell))) )
           else True)
   proof −
    fix j :: nat
```

assume *a*: $j \leq$ *nlength ll*
show *nnth ll j* =
(*if P* (*nnth nell j*) *then nnth l* (*the-enat* (*nlength* (*nfilter P* (*ntaken j nell*))))
else True)
proof −
have *20*: *nnth* (*cnt-in-nellist nell P n*) *j* =
(*if* ∀*ja*≤*j*. ¬ *P* (*nnth nell ja*) *then* (*nnth nell j*, *n*)
else (*nnth nell j*, *n* + *the-enat* (*nlength* (*nfilter P* (*ntaken j nell*))) + *1*))
using *a cnt-in-nellist-nnth*[*of j nell P n*] **using** *1* **by** *auto*
have *22*: (λ(*x*, *k*). *if P x then nnth l* (*k* − *n* − *1*) *else True*) (*nnth* (*cnt-in-nellist nell P n*) *j*) =
(*if P* (*nnth nell j*) *then nnth l* (*the-enat* (*nlength* (*nfilter P* (*ntaken j nell*))))
else True)
by (*simp add*: *20*)
show *?thesis*
using *22 a assms*(*2*) **by** *force*
qed
qed
have *3*: *nlength l* = *nlength* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell ll*)))
using *assms* **by** *auto*
have *4*: ∃ *x*∈*nset* (*nzip nell ll*). (*P* ∘ *fst*) *x*
using *assms*
by (*metis* (*no-types*, *lifting*) *1 comp-apply imageE nellist.set-map nmap-fst-nzip-a*)
have *5*: ⋀*j* . *j* ≤ *nlength l* ⟹
nnth l j =
nnth (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell ll*))) *j*
proof −
fix *j* :: *nat*
assume *a*: *j* ≤ *nlength l*
show *nnth l j* = *nnth* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell ll*))) *j*
proof−
have *6*: *nnth* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell ll*))) *j* =
snd (*nnth* (*nfilter* (*P* ∘ *fst*) (*nzip nell ll*)) *j*)
using *nnth-nmap*[*of j* (*nfilter* (*P*∘*fst*) (*nzip nell ll*)) *snd*] *a 3* **by** *auto*
have *7*: *enat j* ≤ *nlength* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*))
using *a 4 3* **by** (*simp add*: *nkfilter-nlength*)
have *110*: (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*) ≤ *nlength nell*
using *nkfilter-upperbound*[*of* (*nzip nell ll*) (*P* ∘ *fst*) *j 0*]
using *4 7 zero-enat-def* **by** *fastforce*
have *8*: (*nnth* (*nfilter* (*P* ∘ *fst*) (*nzip nell ll*)) *j*) =
nnth (*nzip nell ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j* − *0*)
using *assms nkfilter-nfilter*[*of* (*nzip nell ll*) *P*∘*fst j 0*]
using *4 7* **by** *argo*
have *9*: *snd* (*nnth* (*nzip nell ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j* − *0*)) =
(*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*))
by (*simp add*: *1 110 nnth-nzip*)
have *10*: (*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*)) =
(*if P* (*nnth nell* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*))
then nnth l (*the-enat* (*nlength* (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*)
nell)))) *else True*)
using *2*[*of* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*)]

```
      using 1 110 by auto
    have 11: P (nnth nell (nnth (nkfilter (P ∘ fst) 0 (nzip nell ll)) j))
      by (metis 1 110 3 4 8 a comp-eq-dest-lhs minus-nat.diff-0 nkfilter-nnth-aa nlength-nmap nnth-nzip
split-pairs)
    have 12: (nnth ll (nnth (nkfilter (P ∘ fst) 0 (nzip nell ll)) j)) =
          nnth l (the-enat (nlength (nfilter P (ntaken (nnth (nkfilter (P ∘ fst) 0 (nzip nell ll)) j) nell))))
    using 10 11 by auto
    have 13: (P ∘ fst) (nnth (nzip nell ll) (nnth (nkfilter (P ∘ fst) 0 (nzip nell ll)) j))
      by (metis 4 7 8 diff-zero nkfilter-nlength          nkfilter-nnth-aa)
    have 14: ntaken j (nfilter (P ∘ fst) (nzip nell ll)) =
          nfilter (P ∘ fst) (ntaken (nnth (nkfilter (P ∘ fst) 0 (nzip nell ll)) j) (nzip nell ll))
     using 13 7 nfilter-nkfilter-ntaken by blast
    have 15: (nfilter P (ntaken (nnth (nkfilter (P ∘ fst) 0 (nzip nell ll)) j) nell)) =
           ntaken j (nfilter P nell)
           using 1 14 nmap-fst-nzip-a[of nell ll]
           by (metis nfilter-nmap ntaken-nmap)
    have 16: j ≤ nlength (nfilter P nell)
      by (metis 1 3 a nlength-nfilter-nmap nlength-nmap nmap-fst-nzip-a)
    have 17: (the-enat (nlength (ntaken j (nfilter P nell)) )) = j
          using  ntaken-nlength[of j (nfilter P nell) ]
          by (simp add: 16)
    show ?thesis
    using 12 15 17 6 8 9 by presburger
   qed
  qed
 show ?thesis
 by (simp add: 3 5 nellist-eq-nnth-eq)
qed

lemma cnt-in-nellist-nfilter-var:
assumes ∃ x ∈ nset nell. P x
      ll = nmap (λ (x,k). if P x then (nnth l (k − n − 1)) else X ) (cnt-in-nellist nell P n)
      nlength l = nlength (nmap snd (nfilter (P∘fst) (nzip nell ll)))
shows  l = nmap snd (nfilter (P∘fst) (nzip nell ll))
proof −
 have 1: nlength ll = nlength nell
   using assms    by (simp add: cnt-in-nellist-nlength)
 have 2: ⋀j . j ≤ nlength ll ⟹
          nnth ll j =
          (if P (nnth nell j) then
          nnth l ( the-enat (nlength (nfilter P (ntaken j nell))) )
           else X)
  using  assms cnt-in-nellist-nnth[of - nell P n]
    apply auto
    using 1 apply force
     by (simp add: cnt-in-nellist-nlength)
 have 3: nlength l = nlength (nmap snd (nfilter (P∘fst) (nzip nell ll)))
    using assms by auto
 have 4: ∃ x∈nset (nzip nell ll). (P ∘ fst) x
  using assms
```

by (*metis* (*no-types*, *lifting*) *1 comp-apply imageE nellist.set-map nmap-fst-nzip-a*)
have *5*: $\bigwedge j$. $j \leq$ *nlength l* $\Longrightarrow$
nnth l j =
nnth (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell ll*))) *j*
proof −
fix *j* :: *nat*
assume *a*: $j \leq$ *nlength l*
show *nnth l j* = *nnth* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell ll*))) *j*
proof−
have *6*: *nnth* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip nell ll*))) *j* =
snd (*nnth* (*nfilter* (*P* ∘ *fst*) (*nzip nell ll*)) *j*)
using *nnth-nmap*[*of j* (*nfilter* (*P*∘*fst*) (*nzip nell ll*)) *snd*] *a 3* **by** *auto*
have *7*: *enat j* ≤ *nlength* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*))
using *a 4 3* **by** (*simp add*: *nkfilter-nlength*)
have *110*: (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*) ≤ *nlength nell*
using *nkfilter-upperbound*[*of* (*nzip nell ll*) (*P* ∘ *fst*) *j 0*]
using *4 7 zero-enat-def* **by** *fastforce*
have *8*: (*nnth* (*nfilter* (*P* ∘ *fst*) (*nzip nell ll*)) *j*) =
nnth (*nzip nell ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j* − *0*)
using *assms nkfilter-nfilter*[*of* (*nzip nell ll*) *P*∘*fst j 0*]
using *4 7* **by** *argo*
have *9*: *snd* (*nnth* (*nzip nell ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j* − *0*)) =
(*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*))
by (*simp add*: *1 110 nnth-nzip*)
have *10*: (*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*)) =
(*if P* (*nnth nell* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*))
then nnth l (*the-enat* (*nlength* (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*)
nell)))) *else X*)
using *2*[*of* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*)]
by (*metis* (*mono-tags*, *lifting*) *1 4 7 add-0 min.idem nkfilter-upperbound nlength-nzip zero-enat-def*)
have *11*: *P* (*nnth nell* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*))
by (*metis 1 110 3 4 8 a comp-apply minus-nat.diff-0 nkfilter-nnth-aa nlength-nmap nnth-nzip split-pairs*)
have *12*: (*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*)) =
nnth l (*the-enat* (*nlength* (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*) *nell*))))
using *10 11* **by** *auto*
have *13*: (*P* ∘ *fst*) (*nnth* (*nzip nell ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*))
by (*metis 4 7 8 diff-zero nkfilter-nlength nkfilter-nnth-aa*)
have *14*: *ntaken j* (*nfilter* (*P* ∘ *fst*) (*nzip nell ll*)) =
nfilter (*P* ∘ *fst*) (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*) (*nzip nell ll*))
using *13 7 nfilter-nkfilter-ntaken* **by** *blast*
have *15*: (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip nell ll*)) *j*) *nell*)) =
ntaken j (*nfilter P nell*)
using *nmap-fst-nzip-a*[*of nell ll*]
by (*metis 1 14 nfilter-nmap ntaken-nmap*)
have *16*: *j* ≤ *nlength* (*nfilter P nell*)
by (*metis 1 3 a nlength-nfilter-nmap nlength-nmap nmap-fst-nzip-a*)
have *17*: (*the-enat* (*nlength* (*ntaken j* (*nfilter P nell*)))) = *j*
using *ntaken-nlength*[*of j* (*nfilter P nell*)]
by (*simp add*: *16*)
show *?thesis*

```
    using 12 15 17 6 8 9 by presburger
   qed
  qed
 show ?thesis
 by (simp add: 3 5 nellist-eq-nnth-eq)
qed


lemma cnt-in-nellist-nfilter-ndropns:
assumes ∃ x ∈ nset (ndropns nell). P x
       ll = nmap (λ (x,k). if P x then (nnth l (k − n − 1)) else True ) (cnt-in-nellist (ndropns nell) P n)
      nlength l = nlength (nmap snd (nfilter (P∘fst) (nzip (ndropns nell) (ndropns ll))))
shows  l = nmap (λ xx. (nnth xx 0)) (nmap snd (nfilter (P∘fst) (nzip (ndropns nell) (ndropns ll))))
proof −
 have 1: nlength (ndropns nell) = nlength ll
    by (simp add: assms(2) cnt-in-nellist-nlength ndropns-nlength)
 have 2: ⋀j . enat j ≤ nlength ll ⟹
           nnth ll j =
           (if P (nnth (ndropns nell) j) then
           nnth l ( the-enat (nlength (nfilter P (ntaken j (ndropns nell)))) )
            else True)
   proof −
    fix j :: nat
    assume a: j ≤ nlength ll
    show nnth ll j =
         (if P (nnth (ndropns nell) j) then nnth l ( the-enat (nlength (nfilter P (ntaken j (ndropns nell))))
)
           else True)
    proof −
     have 20: nnth (cnt-in-nellist (ndropns nell) P n) j =
           (if ∀ ja≤j. ¬ P (nnth (ndropns nell) ja) then (nnth (ndropns nell) j, n)
            else (nnth (ndropns nell) j, n + the-enat (nlength (nfilter P (ntaken j (ndropns nell)))) + 1))
       using a  cnt-in-nellist-nnth[of j ndropns nell  P n]  using 1 by auto
     have 22: (λ(x, k). if P x then nnth l (k − n − 1) else True) (nnth (cnt-in-nellist (ndropns nell) P n)
j) =
                (if P (nnth (ndropns nell) j) then nnth l ( the-enat (nlength (nfilter P (ntaken j (ndropns
nell)))) )
           else True)
       by (simp add: 20)
     show ?thesis
     using 22 a assms(2) by force
    qed
   qed
 have 3: nlength (ndropns nell) = nlength (ndropns ll)
    using 1      by (simp add: ndropns-nlength)
  have 030: nset (ndropns nell) = { (ndropn k nell) | k. k ≤ nlength nell}
     by (simp add: nset-ndropns)
  have 031: nset (nzip (ndropns nell) (ndropns ll)) =
           { (ndropn k nell, ndropn k ll) | k. k≤ nlength nell }
    using 1 nset-nzip[of (ndropns nell) (ndropns ll) ]
```

ndropns-nnth[*of - nell*] *ndropns-nnth*[*of - ll*] *ndropns-nlength*[*of nell*] *ndropns-nlength*[*of ll*] **apply** *simp*

by *fastforce*
have *4*: $\exists x{\in}nset\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)).\ (P \circ fst)\ x$
using *031* **using** *030 assms(1)* **by** *auto*
have *5*: $\bigwedge j.\ j \le nlength\ l \Longrightarrow$
$nnth\ l\ j =$
$nnth\ (nmap\ (\lambda\ xx.\ (nnth\ xx\ 0))\ (nmap\ snd\ (nfilter\ (P{\circ}fst)\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))))\ j$
proof −
fix j :: *nat*
assume *a*: *enat* $j \le nlength\ l$
show $nnth\ l\ j =$
$nnth\ (nmap\ (\lambda\ xx.\ (nnth\ xx\ 0))\ (nmap\ snd\ (nfilter\ (P{\circ}fst)\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))))\ j$
proof −
have *6*: $nnth\ (nmap\ (\lambda\ xx.\ (nnth\ xx\ 0))\ (nmap\ snd\ (nfilter\ (P{\circ}fst)\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))))\ j =$
$(\lambda\ xx.\ (nnth\ xx\ 0))\ (nnth\ (nmap\ snd\ (nfilter\ (P{\circ}fst)\ (nzip\ (ndropns\ nell)\ (ndropns\ ll))))\ j)$
using *a assms(3)* **by** *auto*
have *61*: $(\lambda\ xx.\ (nnth\ xx\ 0))\ (nnth\ (nmap\ snd\ (nfilter\ (P{\circ}fst)\ (nzip\ (ndropns\ nell)\ (ndropns\ ll))))\ j) =$
$nnth\ (snd\ (nnth\ (nfilter\ (P{\circ}fst)\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j))\ 0$
using *nnth-nmap*[*of j* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))) *snd*]
using *a assms(3)* **by** *auto*
have *7*: $enat\ j \le nlength\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))$
using *a 4 3* **by** (*simp add*: *assms(3) nkfilter-nlength*)
have *8*: $(nnth\ (nfilter\ (P \circ fst)\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j) =$
$nnth\ (nzip\ (ndropns\ nell)\ (ndropns\ ll))\ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j - 0)$
using *assms nkfilter-nfilter*[*of* (*nzip* (*ndropns nell*) (*ndropns ll*)) *P*∘*fst j 0*]
using *4 7* **by** *presburger*
have *81*: $(nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j) \le nlength\ \ (nzip\ (ndropns\ nell)\ (ndropns\ ll))$
by (*metis 4 7 add-0 nkfilter-upperbound zero-enat-def*)
have *82*: $nlength\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)) = nlength\ nell$
using *3* **by** (*simp add*: *ndropns-nlength*)
have *83*: $enat\ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j) \le nlength\ (ndropns\ nell)$
using *81* **by** *auto*
have *84*: $enat\ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j) \le nlength\ ll$
using *1 81 82* **by** *simp*
have *85*: $(nnth\ (nzip\ (ndropns\ nell)\ (ndropns\ ll))\ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j\ )) =$
$(nnth\ (ndropns\ nell)\ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j\ ),$
$(nnth\ (ndropns\ ll)\ \ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j\ )))$
using *nnth-nzip*[*of* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)
ndropns nell ndropns ll]
by (*metis 83 84 ndropns-nlength*)

have *9*: $snd\ (nnth\ (nzip\ (ndropns\ nell)\ (ndropns\ ll))\ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j\ )) =$
$nnth\ (ndropns\ ll)\ (nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ nell)\ (ndropns\ ll)))\ j\ )$
by (*simp add*: *85*)

have *91*: *nnth* (*nnth* (*ndropns ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) *0* =
nnth ll (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)
by (*simp add*: *84 ndropns-nnth*)
have *10*: (*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) =
(*if P* (*nnth* (*ndropns nell*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*))
then nnth l (*the-enat* (*nlength* (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) (*ndropns nell*)))))
else True)
using *2* [*of* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)]
by (*metis 3 83 ndropns-nlength*)
have *11*: *P* (*nnth* (*ndropns nell*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*))
by (*metis* (*mono-tags*, *lifting*) *4 7 8 85 comp-apply diff-zero fst-conv nkfilter-nlength nkfilter-nnth-aa*)

have *12*: (*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) =
nnth l (*the-enat* (*nlength* (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) (*ndropns nell*)))))
using *11 10* **by** *auto*
have *13*: (*P* ∘ *fst*) (*nnth* (*nzip* (*ndropns nell*) (*ndropns ll*)) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*))
by (*simp add*: *11 85*)
have *14*: *ntaken j* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))) =
nfilter (*P* ∘ *fst*) (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) (*nzip* (*ndropns nell*) (*ndropns ll*)))
using *nfilter-nkfilter-ntaken*[*of* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*)) *j*]
using *13 7* **by** *blast*
have *15*: (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) (*ndropns nell*))) =
ntaken j (*nfilter P* (*ndropns nell*))
using *1 14*
by (*metis* (*no-types*, *opaque-lifting*) *ndropns-nlength nfilter-nmap nmap-fst-nzip-a ntaken-nmap*)
have *16*: *j* ≤ *nlength* (*nfilter P* (*ndropns nell*))
by (*metis 1 a assms*(*3*) *ndropns-nlength nlength-nfilter-nmap nlength-nmap nmap-fst-nzip-a*)
have *17*: (*the-enat* (*nlength* (*ntaken j* (*nfilter P* (*ndropns nell*))))) = *j*
by (*simp add*: *16*)
show *?thesis*
using *12 15 17 6 61 8 9 91* **by** *auto*
qed
qed
have *18*: *nlength l* = *nlength* (*nmap* (λ *xx*. (*nnth xx 0*)) (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*)))))
by (*simp add*: *assms*(*3*))

show *?thesis* **using** *18 5 nellist-eq-nnth-eq* **by** *blast*
qed

lemma *cnt-in-nellist-nfilter-ndropns-var*:
assumes ∃ *x* ∈ *nset* (*ndropns nell*). *P x*
ll = *nmap* (λ (*x*,*k*). *if* (*P*∘*fst*) *x then* (*nnth l* (*k* − *n* − *1*)) *else* (*snd x*)) (*cnt-in-nellist* (*nzip* (*ndropns nell*) *l1*) (*P*∘*fst*) *n*)
nlength l = *nlength* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))))

nlength nell = nlength l1
shows *l = nmap (λ xx. (nnth xx 0)) (nmap snd (nfilter (P∘fst) (nzip (ndropns nell) (ndropns ll))))*
proof −
have *1*: *nlength nell = nlength ll*
by (*simp add*: *assms cnt-in-nellist-nlength ndropns-nlength*)
have *01*: *nlength (nzip (ndropns nell) l1) = nlength nell*
using *assms* **by** (*simp add*: *ndropns-nlength*)
have *02*: *nset (nzip (ndropns nell) l1) = { x | x. (∃ i. i ≤ nlength nell ∧ (fst x) = (ndropn i nell) ∧ (snd x) = (nnth l1 i)) }*
using *nset-nzip*[*of ndropns nell l1*] *assms 01 ndropns-nnth*[*of - nell*] *ndropns-nnth*[*of - l1*]
ndropns-nlength[*of nell*] *ndropns-nlength*[*of l1*] **apply** *simp*
by *fastforce*
have *03*: *∃ x ∈ nset (nzip (ndropns nell) l1). (P∘fst) x*
using *02 assms ndropns-nlength*[*of nell*] *ndropns-nlength*[*of l1*] **apply** *simp*
by (*metis in-nset-ndropns*)
have *2*: *⋀j . enat j ≤ nlength ll ⟹*
nnth ll j =
(if P (nnth (ndropns nell) j) then
nnth l (the-enat (nlength (nfilter P (ntaken j (ndropns nell)))))
else (nnth l1 j))
proof −
fix *j* :: *nat*
assume *aa*: *enat j ≤ nlength ll*
show *nnth ll j =*
(if P (nnth (ndropns nell) j) then
nnth l (the-enat (nlength (nfilter P (ntaken j (ndropns nell)))))
else (nnth l1 j))
proof −
have *020*: *nnth (cnt-in-nellist (nzip (ndropns nell) l1) (P ∘ fst) n) j =*
(if ∀ja≤j. ¬ (P ∘ fst) (nnth (nzip (ndropns nell) l1) ja) then (nnth (nzip (ndropns nell) l1) j, n)
else (nnth (nzip (ndropns nell) l1) j, n + the-enat (nlength (nfilter (P ∘ fst) (ntaken j (nzip (ndropns nell) l1)))) + 1))
using *assms cnt-in-nellist-nnth*[*of j nzip (ndropns nell) l1 P∘ fst n*]
using *01 1 aa* **by** *presburger*
have *021*: *∃ i. i ≤ nlength nell ∧ (P ∘ fst) ((ndropn i nell), (nnth l1 i))*
using *03* **using** *02* **by** *auto*
have *022*: *nnth ll j =*
nnth (nmap (λ (x,k). if (P∘fst) x then (nnth l (k − n − 1)) else (snd x)) (cnt-in-nellist (nzip (ndropns nell) l1) (P∘fst) n)) j
using *assms* **by** *blast*
have *023*: *nnth (nmap (λ (x,k). if (P∘fst) x then (nnth l (k − n − 1)) else (snd x)) (cnt-in-nellist (nzip (ndropns nell) l1) (P∘fst) n)) j =*
(λ (x,k). if (P∘fst) x then (nnth l (k − n − 1)) else (snd x)) (nnth (cnt-in-nellist (nzip (ndropns nell) l1) (P∘fst) n) j)
by (*metis* (*no-types*, *lifting*) *01 1 aa cnt-in-nellist-nlength nnth-nmap*)
have *0230*: *(nnth (cnt-in-nellist (nzip (ndropns nell) l1) (P∘fst) n) j) =*
(nnth (nzip (ndropns nell) l1) j,
(if ∀ja≤j. ¬ (P ∘ fst) (nnth (nzip (ndropns nell) l1) ja) then n else n + the-enat (nlength (nfilter (P ∘ fst) (ntaken j (nzip (ndropns nell) l1)))) + 1))

using *020* **by** *argo*

have *024*: (λ (*x*,*k*). *if* (*P*∘*fst*) *x then* (*nnth l* (*k* − *n* − *1*)) *else* (*snd x*)) (*nnth* (*cnt-in-nellist* (*nzip* (*ndropns nell*) *l1*) (*P*∘*fst*) *n*) *j*) =
(λ (*x*,*k*). *if* (*P*∘*fst*) *x then* (*nnth l* (*k* − *n* − *1*)) *else* (*snd x*)) ((*if* ∀*ja*≤*j*. ¬ (*P* ∘ *fst*) (*nnth* (*nzip* (*ndropns nell*) *l1*) *ja*) *then* (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*, *n*)
else (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*, *n* + *the-enat* (*nlength* (*nfilter* (*P* ∘ *fst*) (*ntaken j* (*nzip* (*ndropns nell*) *l1*)))) + *1*)))
using *020* **by** *simp*

have *0240*: (λ (*x*,*k*). *if* (*P*∘*fst*) *x then* (*nnth l* (*k* − *n* − *1*)) *else* (*snd x*)) (*nnth* (*cnt-in-nellist* (*nzip* (*ndropns nell*) *l1*) (*P*∘*fst*) *n*) *j*) =
(*if* (*P*∘*fst*) (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*) *then*
(*if* ∀*ja*≤*j*. ¬ (*P* ∘ *fst*) (*nnth* (*nzip* (*ndropns nell*) *l1*) *ja*) *then* (*nnth l* (*n* − *n* − *1*))
else (*nnth l* (*n* + *the-enat* (*nlength* (*nfilter* (*P* ∘ *fst*) (*ntaken j* (*nzip* (*ndropns nell*) *l1*)))) + *1* − *n* − *1*)))
else snd (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*))
by (*simp add*: *0230*)

have *0241*: (*P*∘*fst*) (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*) = *P* (*nnth* (*ndropns nell*) *j*)
by (*metis 1 aa assms*(*4*) *comp-apply fst-conv ndropns-nlength nnth-nzip*)

have *0242*: *P* (*nnth* (*ndropns nell*) *j*) ⟹¬(∀*ja*≤*j*. ¬ (*P* ∘ *fst*) (*nnth* (*nzip* (*ndropns nell*) *l1*) *ja*))
using *0241* **by** *auto*

have *0243*: *P* (*nnth* (*ndropns nell*) *j*) ⟹ (*n* + *the-enat* (*nlength* (*nfilter* (*P* ∘ *fst*) (*ntaken j* (*nzip* (*ndropns nell*) *l1*)))) + *1* − *n* − *1*) =
(*the-enat* (*nlength* (*nfilter P* (*ntaken j* (*ndropns nell*)))))
using *assms*(*4*) **apply** *simp*
by (*metis assms*(*4*) *ndropns-nlength nlength-nfilter-nmap nmap-fst-nzip-a ntaken-nmap*)

have *0244*: *snd* (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*) = (*nnth l1 j*)
by (*metis 1 aa assms*(*4*) *ndropns-nlength nnth-nzip snd-conv*)

have *025*: (λ (*x*,*k*). *if* (*P*∘*fst*) *x then* (*nnth l* (*k* − *n* − *1*)) *else* (*snd x*)) ((*if* ∀*ja*≤*j*. ¬ (*P* ∘ *fst*) (*nnth* (*nzip* (*ndropns nell*) *l1*) *ja*) *then* (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*, *n*)
else (*nnth* (*nzip* (*ndropns nell*) *l1*) *j*, *n* + *the-enat* (*nlength* (*nfilter* (*P* ∘ *fst*) (*ntaken j* (*nzip* (*ndropns nell*) *l1*)))) + *1*))) =
(*if P* (*nnth* (*ndropns nell*) *j*) *then*
nnth l (*the-enat* (*nlength* (*nfilter P* (*ntaken j* (*ndropns nell*)))))
else (*nnth l1 j*))
using *024 0240 0241 0242 0243 0244* **by** *argo*

show *?thesis*
using *023 024 025 assms*(*2*) **by** *presburger*
qed
qed

have *3*: *nlength* (*ndropns nell*) = *nlength* (*ndropns ll*)
using *1* **by** (*simp add*: *ndropns-nlength*)

have *030*: *nset* (*ndropns nell*) = { (*ndropn k nell*) | *k*. *k* ≤ *nlength nell*}
by (*simp add*: *nset-ndropns*)

have *031*: *nset* (*nzip* (*ndropns nell*) (*ndropns ll*)) =
{ (*ndropn k nell*, *ndropn k ll*) | *k*. *k*≤ *nlength nell* }
using *1 nset-nzip*[*of* (*ndropns nell*) (*ndropns ll*)] *ndropns-nnth*[*of - nell*] *ndropns-nnth*[*of - ll*]
ndropns-nlength[*of nell*] *ndropns-nlength*[*of ll*] **apply** *auto* **by** *metis*

have *4*: ∃ *x*∈*nset* (*nzip* (*ndropns nell*) (*ndropns ll*)). (*P* ∘ *fst*) *x*
using *031* **using** *030 assms*(*1*) **by** *auto*

have *5*: $\bigwedge j.\ j \leq nlength\ l \Longrightarrow$
nnth l j =
nnth (*nmap* (λ *xx*. (*nnth xx 0*)) (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))))) *j*
proof −
fix *j* :: *nat*
assume *a*: *enat j* ≤ *nlength l*
show *nnth l j* =
nnth (*nmap* (λ *xx*. (*nnth xx 0*)) (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))))) *j*
proof −
have *6*: *nnth* (*nmap* (λ *xx*. (*nnth xx 0*)) (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))))) *j* =
(λ *xx*. (*nnth xx 0*)) (*nnth* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*)))) *j*)
using *a assms*(*3*) **by** *auto*
have *61*: (λ *xx*. (*nnth xx 0*)) (*nnth* (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*)))) *j*) =
nnth (*snd* (*nnth* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) *0*
using *nnth-nmap*[*of j* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))) *snd*]
using *a assms*(*3*) **by** *auto*
have *7*: *enat j* ≤ *nlength* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*)))
using *a 4 3* **by** (*simp add*: *assms*(*3*) *nkfilter-nlength*)
have *8*: (*nnth* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) =
nnth (*nzip* (*ndropns nell*) (*ndropns ll*)) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j* − *0*)
using *assms nkfilter-nfilter*[*of* (*nzip* (*ndropns nell*) (*ndropns ll*)) *P*∘*fst j 0*]
using *4 7* **by** *presburger*
have *81*: (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) ≤ *nlength* (*nzip* (*ndropns nell*) (*ndropns ll*))
by (*metis 4 7 add-0 nkfilter-upperbound zero-enat-def*)
have *82*: *nlength* (*nzip* (*ndropns nell*) (*ndropns ll*)) = *nlength nell*
using *3* **by** (*simp add*: *ndropns-nlength*)
have *83*: *enat* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) ≤ *nlength* (*ndropns nell*)
using *81* **by** *auto*
have *84*: *enat* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) ≤ *nlength ll*
using *1 81 82* **by** *argo*
have *85*: (*nnth* (*nzip* (*ndropns nell*) (*ndropns ll*)) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) =
(*nnth* (*ndropns nell*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*),
(*nnth* (*ndropns ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)))
using *nnth-nzip*[*of* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)
ndropns nell ndropns ll]
by (*metis 83 84 ndropns-nlength*)

have *9*: *snd* (*nnth* (*nzip* (*ndropns nell*) (*ndropns ll*)) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) =
nnth (*ndropns ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)
by (*simp add*: *85*)
have *91*: *nnth* (*nnth* (*ndropns ll*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) *0* =
nnth ll (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)
by (*simp add*: *84 ndropns-nnth*)
have *10*: (*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) =
(*if P* (*nnth* (*ndropns nell*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*))

then nnth l (the-enat (nlength (nfilter P (ntaken (nnth (nkfilter (P ∘ fst) 0 (nzip (ndropns nell) (ndropns ll))) j) (ndropns nell)))))
else (nnth l1 j))
using *2* [*of* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)]
by (*metis* (*mono-tags*, *lifting*) *4 7 8 84 85 comp-apply diff-zero fst-conv nkfilter-nlength nkfilter-nnth-aa*)

have *11*: *P* (*nnth* (*ndropns nell*) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*))
by (*metis* (*mono-tags*, *lifting*) *4 7 8 85 comp-apply diff-zero fst-conv nkfilter-nlength nkfilter-nnth-aa*)

have *12*: (*nnth ll* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*)) =
nnth l (*the-enat* (*nlength* (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) (*ndropns nell*)))))
using *11 10* **by** *auto*
have *13*: (*P* ∘ *fst*) (*nnth* (*nzip* (*ndropns nell*) (*ndropns ll*)) (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*))
by (*simp add*: *11 85*)
have *14*: *ntaken j* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*))) =
nfilter (*P* ∘ *fst*) (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) (*nzip* (*ndropns nell*) (*ndropns ll*)))
using *nfilter-nkfilter-ntaken*[*of* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*)) *j*]
using *13 7* **by** *blast*
have *15*: (*nfilter P* (*ntaken* (*nnth* (*nkfilter* (*P* ∘ *fst*) *0* (*nzip* (*ndropns nell*) (*ndropns ll*))) *j*) (*ndropns nell*))) =
ntaken j (*nfilter P* (*ndropns nell*))
using *1 14*
by (*metis* (*no-types*, *opaque-lifting*) *ndropns-nlength nfilter-nmap nmap-fst-nzip-a ntaken-nmap*)
have *16*: *j* ≤ *nlength* (*nfilter P* (*ndropns nell*))
by (*metis 1 a assms*(*3*) *ndropns-nlength nlength-nfilter-nmap nlength-nmap nmap-fst-nzip-a*)
have *17*: (*the-enat* (*nlength* (*ntaken j* (*nfilter P* (*ndropns nell*))))) = *j*
by (*simp add*: *16*)
show *?thesis*
using *12 15 17 6 61 8 9 91* **by** *auto*
qed
qed
have *18*: *nlength l* = *nlength* (*nmap* (λ *xx*. (*nnth xx 0*)) (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns nell*) (*ndropns ll*)))))
by (*simp add*: *assms*(*3*))

show *?thesis* **using** *18 5 nellist-eq-nnth-eq* **by** *blast*
qed

similar

lemma *similar-defs* [*simp*]:
similar σ0 n σ1 ⟷ (*nlength σ0* = *nlength σ1* ∧
(∀ *k p* . *k*≤ *nlength σ0* ⟶ (*p* |∈| (*nnth σ0 k*) ∧ *p* ≠ *n*) ⟷ (*p* |∈| (*nnth σ1 k*) ∧ *p* ≠ *n*)))
unfolding *similar-def* **by** *auto*

lemma *similar-defs-alt*:

$(nlength\ \sigma 0 = nlength\ \sigma 1 \wedge (\forall\ k\ .\ k \leq nlength\ \sigma 0 \longrightarrow\ (nnth\ \sigma 0\ k)\ |-|\ \{|n|\} = (nnth\ \sigma 1\ k) - \{|n|\}))$
$\longleftrightarrow similar\ \sigma 0\ n\ \sigma 1$
unfolding *similar-def* **by** *auto*

lemma *similars-defs* [*simp*]:
$similars\ \sigma 0\ A\ \sigma 1\ \longleftrightarrow\ (nlength\ \sigma 0 = nlength\ \sigma 1 \wedge$
$(\forall\ k\ p.\ k \leq nlength\ \sigma 0 \longrightarrow\ p\ |\notin|\ A \longrightarrow (\ p\ |\in|\ (nnth\ \sigma 0\ k) \longleftrightarrow p\ |\in|\ (nnth\ \sigma 1\ k))))$
unfolding *similars-def* **by** *auto*

lemma *similars-defs-alt*:
$similars\ \sigma 0\ A\ \sigma 1 \longleftrightarrow\ (nlength\ \sigma 0 = nlength\ \sigma 1 \wedge (\forall\ k.\ k \leq nlength\ \sigma 0 \longrightarrow\ (nnth\ \sigma 0\ k) - A = (nnth\ \sigma 1\ k) - A))$
unfolding *similars-def* **by** *auto*

lemma *similar-similars*:
$similar\ \sigma 0\ n\ \sigma 1 = similars\ \sigma 0\ \{|n|\}\ \sigma 1$
using *similar-defs-alt similars-defs-alt* **by** *auto*

lemma *similars-mono*:
assumes $A\ |\subseteq|\ B$
$similars\ \sigma 0\ A\ \sigma 1$
shows $similars\ \sigma 0\ B\ \sigma 1$
using *assms*
by *fastforce*

lemma *similar-refl*:
$similar\ \sigma\ n\ \sigma$
by *simp*

lemma *similar-commute*:
$similar\ \sigma\ n\ \sigma'\ = similar\ \sigma'\ n\ \sigma$
by *fastforce*

lemma *similar-trans*:
assumes $similar\ \sigma\ n\ \sigma'$
$similar\ \sigma'\ n\ \sigma''$
shows $similar\ \sigma\ n\ \sigma''$
using *assms* **by** *simp*

lemma *similars-refl*:
$similars\ \sigma\ A\ \sigma$
by *simp*

lemma *similars-trans*:
assumes $similars\ \sigma\ A\ \sigma'$
$similars\ \sigma'\ A\ \sigma''$
shows $similars\ \sigma\ A\ \sigma''$

using *assms* **by** *simp*

upd

lemma *upd-NCons*:
upd (*NCons x xs*) *n* (*NCons y ys*) =
(*if y then NCons* (*finsert n x*) (*upd xs n ys*) *else NCons* (*x* |−| {|*n*|}) (*upd xs n ys*))
unfolding *upd-def*
by *simp*

lemma *upd-NNil*:
upd (*NNil a*) *n* (*NNil b*) =
(*if b then* (*NNil* (*finsert n a*)) *else* (*NNil* (*a* |−| {|*n*|})))
unfolding *upd-def*
by (*simp add*: *nfirst-eq-nnth-zero nnth-NNil*)

lemma *nnth-upd*:
assumes *nlength* σ = *nlength l*
(*enat k*) ≤ *nlength* σ
shows (*nnth* (*upd* σ *n l*) *k*) = (*if* (*nnth l k*) *then* (*finsert n* (*nnth* σ *k*)) *else* (*nnth* σ *k*) |−| {|*n*|})
using *assms*
unfolding *upd-def*
by (*simp add*: *nnth-nzip*)

lemma *nlength-upd*:
assumes *nlength* σ = *nlength l*
shows *nlength* ((*upd* σ *n l*)) = *nlength* σ
using *assms*
unfolding *upd-def* **by** *simp*

lemma *upd-swap*:
assumes *nlength* σ = *nlength l1*
nlength σ = *nlength l2*
x1 ≠ *x2*
shows *upd* (*upd* σ *x1 l1*) *x2 l2* = (*upd* (*upd* σ *x2 l2*) *x1 l1*)
proof −
have *1*: *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) = *nlength* (*upd* (*upd* σ *x2 l2*) *x1 l1*)
using *assms nlength-upd* **by** *auto*
have *2*: ⋀ *i*. *i* ≤ *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) ⟹
(*nnth* (*upd* (*upd* σ *x1 l1*) *x2 l2*) *i*) =
(*if* (*nnth l2 i*) *then finsert x2* (*nnth* (*upd* σ *x1 l1*) *i*) *else* (*nnth* (*upd* σ *x1 l1*) *i*) |−| {|*x2*|})
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *3*: ⋀ *i*. *i* ≤ *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) ⟹
(*nnth* (*upd* (*upd* σ *x2 l2*) *x1 l1*) *i*) =
(*if* (*nnth l1 i*) *then finsert x1* (*nnth* (*upd* σ *x2 l2*) *i*) *else* (*nnth* (*upd* σ *x2 l2*) *i*) |−| {|*x1*|})
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *4*: ⋀ *i*. *i* ≤ *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) ⟹

(*if* (*nnth l2 i*) *then finsert x2* (*nnth* (*upd* σ *x1 l1*) *i*) *else* (*nnth* (*upd* σ *x1 l1*) *i*) |−| {|*x2*|}) =
(*if* (*nnth l1 i*) *then finsert x1* (*nnth* (*upd* σ *x2 l2*) *i*) *else* (*nnth* (*upd* σ *x2 l2*) *i*) |−| {|*x1*|})
using *assms nnth-upd nlength-upd* **by** *auto*
show *?thesis*
by (*simp add*: *1 2 3 4 nellist-eq-nnth-eq*)
qed

lemma *upd-absorb*:
assumes *nlength* σ = *nlength l1*
nlength σ = *nlength l2*
x1 = *x2*
shows *upd* (*upd* σ *x1 l1*) *x2 l2* = (*upd* σ *x2 l2*)
proof −
have *1*: *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) = *nlength* (*upd* σ *x2 l2*)
using *assms nlength-upd* **by** *auto*
have *2*: $\bigwedge$ *i*. *i* ≤ *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) ⟹
(*nnth* (*upd* (*upd* σ *x1 l1*) *x2 l2*) *i*) =
(*if* (*nnth l2 i*) *then finsert x2* (*nnth* (*upd* σ *x1 l1*) *i*) *else* (*nnth* (*upd* σ *x1 l1*) *i*) |−| {|*x2*|})
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *3*: $\bigwedge$ *i*. *i* ≤ *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) ⟹
(*nnth* (*upd* σ *x2 l2*) *i*) =
(*if* (*nnth l2 i*) *then finsert x2* (*nnth* σ *i*) *else* (*nnth* σ *i*) |−| {|*x2*|})
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *4*: $\bigwedge$ *i*. *i* ≤ *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) ⟹
(*if* (*nnth l2 i*) *then finsert x2* (*nnth* (*upd* σ *x1 l1*) *i*) *else* (*nnth* (*upd* σ *x1 l1*) *i*) |−| {|*x2*|}) =
(*if* (*nnth l2 i*) *then finsert x2* (*nnth* σ *i*) *else* (*nnth* σ *i*) |−| {|*x2*|})
using *assms nnth-upd nlength-upd* **by** *auto*
show *?thesis*
by (*simp add*: *1 2 3 4 nellist-eq-nnth-eq*)
qed

lemma *upd-nappend*:
assumes *nlength* σ*1* = *nlength l1*
nlength σ*2* = *nlength l2*
shows *upd* (*nappend* σ*1* σ*2*) *x* (*nappend l1 l2*) =
(*nappend* (*upd* σ*1 x l1*) (*upd* σ*2 x l2*))
using *assms* **unfolding** *upd-def*
by (*simp add*: *nmap-nappend-distrib nzip-nappend*)

lemma *ntaken-upd*:
assumes *nlength* σ = *nlength l*
j ≤ *nlength* σ
shows *ntaken j* (*upd* σ *x l*) = (*upd* (*ntaken j* σ) *x* (*ntaken j l*))
using *assms* **unfolding** *upd-def*
by (*simp add*: *ntaken-nzip*)

lemma *ndropn-upd*:
assumes *nlength* σ = *nlength l*

$j \leq nlength\ \sigma$
shows $ndropn\ j\ (upd\ \sigma\ x\ l) = (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l))$
using *assms ndropn-nzip upd-def* **by** (*simp add*: *ndropn-nmap*)

lemma *nsubn-upd*:
assumes $nlength\ \sigma = nlength\ l$
$(enat\ j) \leq enat\ k$
$(enat\ k) \leq nlength\ \sigma$
shows $nsubn\ (upd\ \sigma\ x\ l)\ j\ k\ = (upd\ (nsubn\ \ \sigma\ j\ k)\ x\ (nsubn\ \ l\ j\ k))$
proof −
have *1*: $nsubn\ (upd\ \sigma\ x\ l)\ j\ k = ntaken\ (k - j)\ (ndropn\ j\ (upd\ \sigma\ x\ l))$
using *assms* **unfolding** *nsubn-def1* **by** *simp*
have *2*: $(ndropn\ j\ (upd\ \sigma\ x\ l)) = (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l))$
using *assms* **by** (*meson dual-order.trans ndropn-upd*)
have *3*: $ntaken\ (k - j)\ (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l)) =$
$(upd\ (ntaken\ (k{-}j)\ (ndropn\ j\ \sigma))\ x\ (ntaken\ (k{-}j)\ (ndropn\ j\ l)))$
using *assms ntaken-upd*[*of* $(ndropn\ j\ \sigma)\ (ndropn\ j\ l)\ (k - j)\ x$]
by (*metis ndropn-nlength nle-le nlength-upd ntaken-all*)
have *4*: $(upd\ (ntaken\ (k{-}j)\ (ndropn\ j\ \sigma))\ x\ (ntaken\ (k{-}j)\ (ndropn\ j\ l))) =$
$(upd\ (nsubn\ \ \sigma\ j\ k)\ x\ (nsubn\ \ l\ j\ k))$
unfolding *nsubn-def1* **by** *simp*
show *?thesis* **using** *1 2 3 4* **by** *presburger*
qed

lemma *nfirst-upd*:
assumes $nlength\ \sigma = nlength\ l$
shows $nfirst\ (upd\ \sigma\ x\ l) = (if\ nfirst\ l\ then\ finsert\ x\ (nfirst\ \sigma)\ else\ (nfirst\ \sigma)\ |-|\ \{|x|\}\ )$
by (*metis assms nfirst-eq-nnth-zero nnth-upd zero-enat-def zero-le*)

lemma *nlast-upd*:
assumes $nlength\ \sigma = nlength\ l$
$nfinite\ \sigma$
shows $nlast\ (upd\ \sigma\ x\ l) = (if\ nlast\ l\ then\ finsert\ x\ (nlast\ \sigma)\ else\ (nlast\ \sigma)\ |-|\ \{|x|\}\ )$
using *assms*
by (*metis dual-order.order-iff-strict nfinite-conv-nlength-enat nlength-upd nnth-nlast nnth-upd the-enat.simps*)

lemma *upd-nfuse*:
assumes $nlength\ \sigma 1 = nlength\ l1$
$nlength\ \sigma 2 = nlength\ l2$
$nfirst\ \sigma 2 = nlast\ \sigma 1$
$nfinite\ \sigma 1$
$nfirst\ l2 = nlast\ l1$
shows $upd\ (nfuse\ \sigma 1\ \sigma 2)\ x\ (nfuse\ l1\ l2) =$
$(nfuse\ (upd\ \sigma 1\ x\ l1)\ (upd\ \sigma 2\ x\ l2))$
using *assms* **unfolding** *upd-def* **by** (*simp add*: *nfuse-nlength nmap-nfuse nzip-nfuse*)

lemma *upd-ndropns*:

assumes *nlength σ = nlength l*
shows (*ndropns* (*upd σ n l*)) = (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns σ*) (*ndropns l*)))
proof −
have *1*: *nlength* (*ndropns* (*upd σ n l*)) = *nlength* (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns σ*) (*ndropns l*)))
using *assms* **by** (*simp add*: *ndropns-nlength nlength-upd*)
have *2*: ⋀*j*. *j*≤ *nlength* (*ndropns* (*upd σ n l*)) ⟹
(*nnth* (*ndropns* (*upd σ n l*)) *j*) =
(*ndropn j* (*upd σ n l*))
by (*simp add*: *ndropns-nlength ndropns-nnth*)
have *3*: ⋀*j*. *j*≤ *nlength* (*ndropns* (*upd σ n l*)) ⟹
(*ndropn j* (*upd σ n l*)) =
(*upd* (*ndropn j σ*) *n* (*ndropn j l*))
by (*simp add*: *assms ndropn-upd ndropns-nlength nlength-upd*)
have *4*: ⋀*j*. *j*≤ *nlength* (*ndropns* (*upd σ n l*)) ⟹
(*nnth* (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns σ*) (*ndropns l*))) *j*) =
((*ndropn j* (*upd σ n l*)))
by (*simp add*: *3 assms ndropns-nlength ndropns-nnth nlength-upd nnth-nzip*)
have *5*: (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns σ*) (*ndropns l*))) =
(*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nmap* (λ (*s*,*i*). (*ndropn i σ*, *ndropn i l*)) (*nzip σ* (*niterates Suc 0*))))
using *assms nzip-ndropns-nmap* **by** *fastforce*
have *6*: (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nmap* (λ (*s*,*i*). (*ndropn i σ*, *ndropn i l*)) (*nzip σ* (*niterates Suc 0*)))) =
(*nmap* ((λ(*s*,*ll*). (*upd s n ll*)) ∘ ((λ (*s*,*i*). (*ndropn i σ*, *ndropn i l*)))) (*nzip σ* (*niterates Suc 0*)))
by (*simp add*: *nellist.map-comp*)
have *7*: ((λ(*s*,*ll*). (*upd s n ll*)) ∘ ((λ (*s*,*i*). (*ndropn i σ*, *ndropn i l*)))) =
(λ (*s*,*i*). (*upd* (*ndropn i σ*) *n* (*ndropn i l*)))
by *auto*
show *?thesis*
by (*metis 1 4 ndropns-nlength ndropns-nnth nellist-eq-nnth-eq*)
qed

lemma *upd-nfilter*:
assumes *nlength σ = nlength l*

⋀*i*. *i* ≤ *nlength σ* ⟹
P (*nnth σ i*) = *P* (*nnth* (*upd σ n l*) *i*)
shows *nfilter P* (*upd σ n l*) = *upd* (*nfilter P σ*) *n* (*nmap snd* (*nfilter* (*P*∘ *fst*) (*nzip σ l*)))
proof −
have *1*: *nfilter P* (*upd σ n l*) = *nfilter P* (*nmap* (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|}) (*nzip σ l*))
unfolding *upd-def* **by** *simp*
have *2*: *nfilter P* (*nmap* (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|}) (*nzip σ l*)) =
nmap (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|})
(*nfilter* (*P* ∘ (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|})) (*nzip σ l*))
using *nfilter-nmap*[*of P* (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|}) (*nzip σ l*)]
by *blast*
have *4*: ⋀*xx yy*. (*P* ∘ (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|})) (*xx*, *yy*) =
((*yy* ⟶ *P* (*finsert n xx*)) ∧ (¬ *yy* ⟶ *P* (*xx* |−| {|*n*|})))

by *simp*
have *5*: (*nfilter* (*P* ∘ (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|})) (*nzip* σ *l*)) =
(*nfilter* (λ (*x*,*y*). (*if y then P* (*finsert n x*) *else P* (*x* |−| {|*n*|}))) (*nzip* σ *l*))
using *4* **by** (*metis* (*no-types*, *lifting*) *case-prodE case-prodI2*)
have *50*: ⋀ *xx yy*. (*xx*,*yy*)∈*nset* (*nzip* σ *l*) ⟷
(∃ *i*. *enat i* ≤ *nlength* σ ∧ *xx* = (*nnth* σ *i*) ∧ *yy* = (*nnth l i*))
using *assms*(*1*) *in-nset-conv-nnth*[*of* - *nzip* σ *l*] *nnth-nzip*[*of* - σ *l*] **by** *auto*
have *6*: ⋀ *xx yy*. (*xx*,*yy*)∈*nset* (*nzip* σ *l*) ⟹ (λ (*x*,*y*). (*if y then P* (*finsert n x*) *else P* (*x* |−| {|*n*|}))) (*xx*,*yy*) = (λ(*x*,*y*). *P x*) (*xx*,*yy*)
using *assms*
using *50 nnth-upd* **by** *auto*
have *7*: (*nfilter* (λ (*x*,*y*). (*if y then P* (*finsert n x*) *else P* (*x* |−| {|*n*|}))) (*nzip* σ *l*)) =
(*nfilter* (λ(*x*,*y*). *P x*) (*nzip* σ *l*))
using *6 nfilter-cong*[*of* (*nzip* σ *l*) (*nzip* σ *l*) (λ (*x*,*y*). (*if y then P* (*finsert n x*) *else P* (*x* |−| {|*n*|}))) (λ(*x*,*y*). *P x*)]
by *force*
have *8*: (λ(*x*,*y*). *P x*) = *P* ∘ *fst*
by *auto*
have *90*: (*nmap fst* (*nfilter* (λ(*x*,*y*). *P x*) (*nzip* σ *l*))) =
nfilter P (*nmap fst* (*nzip* σ *l*))
by (*simp add*: *8 nfilter-nmap*)
have *91*: (*nmap fst* (*nzip* σ *l*)) = σ
by (*simp add*: *assms*(*1*) *nmap-fst-nzip-a*)
have *92*: *nfilter P* (*nmap fst* (*nzip* σ *l*)) = *nfilter P* σ
by (*simp add*: *91*)
have *10*: *upd* (*nfilter P* σ) *n* (*nmap snd* (*nfilter* (*P*∘ *fst*) (*nzip* σ *l*))) =
nmap (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|})
(*nzip* (*nfilter P* σ) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*))))
unfolding *upd-def* **by** *simp*
have *11*: (*nzip* (*nfilter P* σ) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)))) =
(*nzip* (*nmap fst* (*nfilter* (*P*∘*fst*) (*nzip* σ *l*))) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*))))
by (*metis 8 90 91*)
have *12*: *nzip* (*nmap fst* (*nfilter* (*P* ∘*fst*) (*nzip* σ *l*))) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*))) =
nmap (λ(*x*, *y*). (*fst x*, *snd y*)) (*nzip* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)) (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)))
using *nzip-nmap*[*of fst* (*nfilter* (*P* ∘*fst*) (*nzip* σ *l*)) *snd* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*))]
by *simp*
have *13*: *nlength* (*nmap* (λ(*x*, *y*). (*fst x*, *snd y*)) (*nzip* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)) (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)))) =
nlength (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*))
by *simp*
have *14*: ⋀*j*. *j*≤ *nlength* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)) ⟹
nnth (*nmap* (λ(*x*, *y*). (*fst x*, *snd y*)) (*nzip* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)) (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)))) *j* =
nnth (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)) *j*
by (*simp add*: *nnth-nzip*)
have *15*: *nmap* (λ(*x*, *y*). (*fst x*, *snd y*)) (*nzip* (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*)) (*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*))) =
(*nfilter* (*P* ∘ *fst*) (*nzip* σ *l*))
by (*metis 13 14 nellist-eq-nnth-eq*)
show *?thesis*
by (*simp add*: *1 10 11 15 2 5 7 8*)

qed

lemma *upd-nfilter-ndropns*:
assumes *nlength* σ = *nlength l*
$\bigwedge i.\ i \leq$ *nlength* $\sigma \Longrightarrow$
P (*ndropn i* σ) = *P* (*ndropn i* (*upd* σ *n l*))
shows (*nfilter P* (*ndropns* (*upd* σ *n l*))) =
nmap (λ(*s*, *ll*). *upd s n ll*) (*nfilter* (*P* $\circ$ (λ(*s*, *ll*). *s*)) (*nzip* (*ndropns* σ) (*ndropns l*)))
proof −
have *1*: (*nfilter P* (*ndropns* (*upd* σ *n l*))) =
(*nfilter P* (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns* σ) (*ndropns l*))))
using *upd-ndropns assms* **by** *auto*
have *2*: (*nfilter P* (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns* σ) (*ndropns l*)))) =
nmap (λ(*s*, *ll*). *upd s n ll*) (*nfilter* (*P* $\circ$ (λ(*s*, *ll*). *upd s n ll*)) (*nzip* (*ndropns* σ) (*ndropns l*)))
using *nfilter-nmap*[*of P* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns* σ) (*ndropns l*))]
by *blast*
have *3*: $\bigwedge$ *xx*. *xx* $\in$ *nset* (*nzip* (*ndropns* σ) (*ndropns l*)) $\Longrightarrow$
(*P* $\circ$ (λ(*s*, *ll*). *upd s n ll*)) *xx* = *P* (*fst xx*)
proof −
fix *xx*
assume *a*: *xx* $\in$ *nset* (*nzip* (*ndropns* σ) (*ndropns l*))
show (*P* $\circ$ (λ(*s*, *ll*). *upd s n ll*)) *xx* = *P* (*fst xx*)
proof −
have *30*: $\exists$ *i*. *i* $\leq$ *nlength* σ $\wedge$ *xx* = (*ndropn i* σ, *ndropn i l*)
using *a assms in-nset-conv-nnth*[*of xx* (*nzip* (*ndropns* σ) (*ndropns l*))]
nnth-nzip[*of - ndropns* σ *ndropns l*] *ndropns-nnth*[*of -* σ] *ndropns-nnth*[*of - l*]
apply *auto* **by** (*metis ndropns-nlength*)
show *?thesis*
using *30 assms*(*1*,*2*) *ndropn-upd* **by** *force*
qed
qed
have *4*: (*nfilter* (*P* $\circ$ (λ(*s*, *ll*). *upd s n ll*)) (*nzip* (*ndropns* σ) (*ndropns l*))) =
(*nfilter* (*P* $\circ$ (λ(*s*, *ll*). *s*)) (*nzip* (*ndropns* σ) (*ndropns l*)))
using *3*
by (*metis* (*mono-tags*, *lifting*) *comp-apply fst-def nfilter-cong*)
show *?thesis*
by (*simp add*: *1 2 4*)
qed

lemma *upd-pifilt*:
assumes *nlength* σ = *nlength l*
$\bigwedge i.\ i \leq$ *nlength* $\sigma \Longrightarrow$
P (*ndropn i* σ) = *P* (*ndropn i* (*upd* σ *n l*))
shows (*pifilt* (*upd* σ *n l*) *P*) =
upd (*pifilt* σ *P*) *n* (*nmap* (λ*s*. *nnth s 0*) (*nmap snd* (*nfilter* (*P*$\circ$*fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))))
proof −
have *1*: (*ndropns* (*upd* σ *n l*)) = (*nmap* (λ(*s*,*ll*). (*upd s n ll*)) (*nzip* (*ndropns* σ) (*ndropns l*)))

using *upd-ndropns assms* **by** *blast*
have *2*: (*nfilter P* (*ndropns* (*upd* σ *n l*))) =
nmap (λ(*s*, *ll*). *upd s n ll*) (*nfilter* (*P* ∘ (λ(*s*, *ll*). *s*)) (*nzip* (*ndropns* σ) (*ndropns l*)))
using *assms upd-nfilter-ndropns* **by** *blast*
have *3*: *nmap* (λ*s*. *nnth s 0*) (*nmap* (λ(*s*, *ll*). *upd s n ll*) (*nfilter* (*P* ∘ (λ(*s*, *ll*). *s*)) (*nzip* (*ndropns* σ) (*ndropns l*)))) =
nmap ((λ*s*. *nnth s 0*)∘(λ(*s*, *ll*). *upd s n ll*)) (*nfilter* (*P* ∘ (λ(*s*, *ll*). *s*)) (*nzip* (*ndropns* σ) (*ndropns l*)))
by (*simp add*: *nellist.map-comp*)
have *4*: *nmap* ((λ*s*. *nnth s 0*)∘(λ(*s*, *ll*). *upd s n ll*)) (*nfilter* (*P* ∘ (λ(*s*, *ll*). *s*)) (*nzip* (*ndropns* σ) (*ndropns l*))) =
nmap ((λ*s*. *nnth s 0*)∘(λ(*s*, *ll*). *upd s n ll*)) (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))
by (*metis fst-def*)
have *5*: *upd* (*pifilt* σ *P*) *n* (*nmap* (λ*s*. *nnth s 0*) (*nmap snd* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))) =
nmap (*upd-fun n*)
(*nzip* (*nmap* (λ*s*. *nnth s 0*) (*nfilter P* (*ndropns* σ)))
(*nmap* (λ*s*. *nnth s 0*) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))))
unfolding *pifilt-def sfxfilt-def upd-fun-def upd-def* **by** *simp*
have *6*: (*nzip* (*nmap* (λ*s*. *nnth s 0*) (*nfilter P* (*ndropns* σ)))
(*nmap* (λ*s*. *nnth s 0*) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))))) =
nmap (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*))) (*nzip* (*nfilter P* (*ndropns* σ)) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))))
using *nzip-nmap* **by** *auto*
have *7*: (*nfilter P* (*ndropns* σ)) = *nmap fst* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))
by (*metis assms*(*1*) *ndropns-nlength nfilter-nmap nmap-fst-nzip-a*)
have *8*: *nmap* (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*)))
(*nzip* (*nfilter P* (*ndropns* σ))
(*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))) =
nmap (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*)))
(*nzip* (*nmap fst* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))
(*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))))
using *7* **by** *auto*
have *9*:
(*nzip* (*nmap fst* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))
(*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))) =
(*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))
proof −
have *90*: *nlength* (*nzip* (*nmap fst* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))
(*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))) =
nlength (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))
by *simp*
have *91*: ⋀*j*. *enat j*≤ *nlength* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))) ⟹
nnth (*nzip* (*nmap fst* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))
(*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))) *j* =
nnth (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))) *j*
by (*simp add*: *nnth-nzip*)
show *?thesis* **by** (*metis 90 91 nellist-eq-nnth-eq*)
qed

have *10*: *nmap* (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*)))
(*nzip* (*nmap fst* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))
(*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))) =
nmap (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*))) (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))
using *9* **by** *auto*
have *11*: *nmap* (*upd-fun n*)
(*nzip* (*nmap* (λ*s*. *nnth s 0*) (*nfilter P* (*ndropns* σ)))
(*nmap* (λ*s*. *nnth s 0*) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))))) =
nmap (*upd-fun n*) (*nmap* (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*))) (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))))
using *6 8 9* **by** *presburger*
have *12*: (*pifilt* (*upd* σ *n l*) *P*) =
nmap ((λ*s*. *nnth s 0*)∘(λ(*s*, *ll*). *upd s n ll*)) (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))
unfolding *pifilt-def sfxfilt-def*
by (*simp add*: *2 3 4*)
have *13*: *nmap* (*upd-fun n*) (*nmap* (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*))) (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))) =
nmap ((*upd-fun n*) ∘ (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*)))) (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*)))
by (*simp add*: *nellist.map-comp*)
have *14*: ⋀*xx yy*. (*xx*, *yy*) ∈ *nset* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))) ⟹
nnth (*nmap* (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|}) (*nzip xx yy*)) *0* =
(λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|}) (*nnth xx 0*, *nnth yy 0*)
using *nnth-nmap nnth-nzip*
using *zero-enat-def*
by (*metis* (*no-types*, *lifting*) *zero-le*)
have *15*: ⋀*xx yy*. (*xx*, *yy*) ∈ *nset* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*ndropns l*))) ⟹
((*upd-fun n*) ∘ (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*)))) (*xx*,*yy*) =
((λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|}) ∘ (λ(*x*, *y*). (*nnth x 0*, *nnth y 0*))) (*xx*, *yy*)
unfolding *upd-fun-def* **by** *simp*

have *16*: ⋀ *xx yy*. (*xx*,*yy*) ∈ *nset* (*nfilter* (*P*∘*fst*) (*nzip* (*ndropns* σ) (*ndropns l*))) ⟹
((λ*s*. *nnth s 0*)∘(λ(*s*, *ll*). *upd s n ll*)) (*xx*,*yy*) = ((*upd-fun n*) ∘ (λ (*x*,*y*). ((*nnth x 0*), (*nnth y 0*)))) (*xx*,*yy*)
using *14 15* **by** (*simp add*: *upd-def*)
show *?thesis*
by (*metis* (*lifting*) *12 13 16 5 6 7 9 nellist.map-cong0 prod.collapse*)
qed

lemma *nkfilter-upd-def1*:
assumes (∃ *x* ∈ *nset* (*nmap fst* (*nzip* (*ndropns* σ) (*ndropns l*))). *P x*)
shows *nkfilter P n* (*nmap fst* (*nzip* (*ndropns* σ) (*ndropns l*))) =
nmap snd (*nfilter* (*P*∘*fst*) (*nzip* (*nmap fst* (*nzip* (*ndropns* σ) (*ndropns l*))) (*niterates Suc n*)))
using *assms* *nkfilter-def1* **by** *metis*

lemma *nkfilter-upd-def2*:
assumes (∃ *x* ∈ *nset* ((*nzip* (*ndropns* σ) (*ndropns l*))). *P* (*fst x*))
shows *nkfilter* (*P*∘*fst*) *n* ((*nzip* (*ndropns* σ) (*ndropns l*))) =
nmap snd (*nfilter* (*P*∘*fst*∘*fst*) (*nzip* ((*nzip* (*ndropns* σ) (*ndropns l*))) (*niterates Suc n*)))
using *assms*

by (*simp add*: *nkfilter-def1*)

lemma *nkfilter-nfilter-upd*:
assumes $\exists\ x \in nset\ (nmap\ fst\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ .\ P\ x$
$i \leq nlength\ (nkfilter\ P\ n\ (nmap\ fst\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))$
shows $(nnth\ (nmap\ fst\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))$
$((nnth\ (nkfilter\ P\ n\ (nmap\ fst\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ i) - n)) =$
$(nnth\ (nfilter\ P\ (nmap\ fst\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ i)$
using *assms nkfilter-nfilter* **by** *metis*

lemma *nkfilter-nfilter-upd-1*:
assumes $\exists\ x \in nset\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ .\ P\ (fst\ x)$
$i \leq nlength\ (nkfilter\ (P \circ fst)\ n\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))$

shows $(nnth\ ((nzip\ (ndropns\ \sigma)\ (ndropns\ l)))$
$((nnth\ (nkfilter\ (P \circ fst)\ n\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ i) - n)) =$
$(nnth\ (nfilter\ (P \circ fst)\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ i)$
using *assms* **by** (*simp add*: *nkfilter-nfilter*)

lemma *nkfilter-nfilter-upd-2*:
assumes $\exists\ x \in nset\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ .\ P\ (fst\ x)$
$i \leq nlength\ (nkfilter\ (P \circ fst)\ n\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))$
$nlength\ l = nlength\ \sigma$
shows $(nnth\ (\ (ndropns\ l))$
$(\ (nnth\ ((nkfilter\ (P \circ fst)\ n\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))))\ i) - n)\ ) =$
$(nnth\ (nmap\ snd\ (nfilter\ (P \circ fst)\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))))\ i)$
proof −
have *1*: $(nnth\ ((nzip\ (ndropns\ \sigma)\ (ndropns\ l)))$
$((nnth\ (nkfilter\ (P \circ fst)\ n\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ i) - n)) =$
$(nnth\ (nfilter\ (P \circ fst)\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ i)$
using *assms* **by** (*simp add*: *nkfilter-nfilter*)
have *20*: $enat\ (nnth\ (nkfilter\ (P \circ fst)\ n\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ i)$
$\leq enat\ n + nlength\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))$
using *nkfilter-upperbound*[*of* $(\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ (P \circ fst)\ i\ n\ ]$
assms **by** *auto*
have *21*: $nnth\ (nkfilter\ (P \circ fst)\ n\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ i - n =$
$nnth\ (nkfilter\ (P \circ fst)\ 0\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ i$
using *nkfilter-nnth-n-zero*[*of* $(\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ (P \circ fst)\ i\ n$]
assms **by** *auto*
have *22*: $nnth\ (nmap\ snd\ (nfilter\ (P \circ fst)\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ i =$
$snd\ (nnth\ (nfilter\ (P \circ fst)\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ i)$
using *nnth-nmap*[*of i* $(nfilter\ (P \circ fst)\ (\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))))\ snd$]
assms **by** (*metis comp-apply nkfilter-nlength*)
have *23*: $enat\ (nnth\ (nkfilter\ (P \circ fst)\ n\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ i - n)$
$\leq nlength\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l))$
using *20*
by (*metis enat.distinct*(*2*) *enat-add-sub-same enat-minus-mono1 idiff-enat-enat*)
have *24*: $nnth\ (nmap\ snd\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))$
$(nnth\ (nkfilter\ (P \circ fst)\ n\ (nzip\ (ndropns\ \sigma)\ (ndropns\ l)))\ i - n) =$

snd (*nnth* (*nzip* (*ndropns* σ) (*ndropns* *l*))
(*nnth* (*nkfilter* (*P* $\circ$ *fst*) *n* (*nzip* (*ndropns* σ) (*ndropns* *l*))) *i* − *n*))
using *23* *nnth-nmap*[*of* ((*nnth* (*nkfilter* (*P*∘*fst*) *n* ((*nzip* (*ndropns* σ) (*ndropns* *l*)))) *i*) − *n*) ((*nzip* (*ndropns* σ) (*ndropns* *l*))) *snd*]
by *auto*
have *2*: (*nnth* (*nmap snd* (*nfilter* (*P*∘*fst*) ((*nzip* (*ndropns* σ) (*ndropns* *l*))))) *i*) =
(*nnth* (*nmap snd* ((*nzip* (*ndropns* σ) (*ndropns* *l*)))) ((*nnth* (*nkfilter* (*P*∘*fst*) *n* ((*nzip* (*ndropns* σ) (*ndropns* *l*)))) *i*) − *n*))
by (*simp add*: *1 22 24*)
have *3*: (*nmap snd* ((*nzip* (*ndropns* σ) (*ndropns* *l*)))) = *ndropns* *l*
by (*simp add*: *assms*(*3*) *ndropns-nlength nmap-snd-nzip*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *exist-expand-upd-list*:
fixes *l* :: *bool nellist*
assumes ∃ *x* ∈ *nset* (*ndropns* σ). *P* (*x*)
nlength (*pifilt* σ *P*) = *nlength* *l*
shows (∃ *ul1*. *nlength ul1* = *nlength* σ ∧
l = *nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) *ul1*)))
proof−
have *1*: *nlength* *l* =
nlength (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns* σ) (*nmap* (λ(*x*, *k*). *if P x then nnth l* (*k* − *0* − *1*) *else True*) (*cnt-in-nellist* (*ndropns* σ) *P 0*)))))
using *assms* **unfolding** *pifilt-def sfxfilt-def*
by (*metis* (*no-types*, *lifting*) *cnt-in-nellist-nlength nfilter-nmap nlength-nmap nmap-fst-nzip-a*)
show *?thesis*
using *assms cnt-in-nellist-nfilter*[*of* (*ndropns* σ) *P*
nmap (λ(*x*, *k*). *if P x then nnth l* (*k* − *0* − *1*) *else True*) (*cnt-in-nellist* (*ndropns* σ) *P 0*) *l 0*] *1*
by (*metis cnt-in-nellist-nlength ndropns-nlength nlength-nmap*)
qed

lemma *pifilt-upd*:
assumes ∃ *x* ∈ *nset* (*ndropns* σ). *P* (*x*)

ll = *nmap* (λ(*x*, *k*). *if P x then nnth l* (*k* − *1*) *else True*) (*cnt-in-nellist* (*ndropns* σ) *P 0*)
nlength (*pifilt* σ *P*) = *nlength* *l*
⋀*i*. *i* ≤ *nlength* σ ⟹
P (*ndropn i* σ) = *P* (*ndropn i* (*upd* σ *n ll*))
shows *upd* (*pifilt* σ *P*) *n l* = *pifilt* (*upd* σ *n ll*) *P*
proof −
have *1*: *nlength* σ = *nlength ll*
using *assms* **by** (*simp add*: *cnt-in-nellist-nlength ndropns-nlength*)
have *2*: *upd* (*pifilt* σ *P*) *n l* =
nmap (λ(*x*, *y*). *if y then finsert n x else x* |−| {|*n*|})
(*nzip* (*nmap* (λ*s*. *nnth s 0*) (*nfilter P* (*ndropns* σ))) *l*)

```
   unfolding upd-def pifilt-def sfxfilt-def by simp
 have 3: pifilt (upd σ n ll ) P =
        nmap (λs. nnth s 0)
            (nfilter P (ndropns (upd σ n ll)))
   unfolding  pifilt-def sfxfilt-def  by simp
 have 4: (ndropns (upd σ n ll)) =
        nmap (λ(s, ll). upd s n ll) (nzip (ndropns σ) (ndropns ll))
   using upd-ndropns    using 1 by auto
 have 5: (nfilter P (nmap (λ(s, ll). upd s n ll) (nzip (ndropns σ) (ndropns ll)))) =
        nmap (λ(s, ll). upd s n ll)(nfilter  (P∘(λ(s, ll). upd s n ll)) (nzip (ndropns σ) (ndropns ll)))
     by (simp add: nfilter-nmap)
 have 6: pifilt (upd σ n ll) P =
          upd (pifilt σ P) n (nmap (λs. nnth s 0) (nmap snd (nfilter (P ∘ fst) (nzip (ndropns σ) (ndropns
ll)))))
   using upd-pifilt[of σ ll P n]
     using 1 assms(4) by auto
 have 7: nlength l = nlength (nmap snd (nfilter (P ∘ fst) (nzip (ndropns σ) (ndropns ll))))
     using assms(3) unfolding pifilt-def sfxfilt-def
     by (metis 1 ndropns-nlength nlength-nfilter-nmap nlength-nmap nmap-fst-nzip-a)
 have 8: l = (nmap (λs. nnth s 0) (nmap snd (nfilter (P ∘ fst) (nzip (ndropns σ) (ndropns ll)))))
   using  cnt-in-nellist-nfilter-ndropns[of σ P ll l 0 ] assms 7 apply simp
   by (metis (no-types, lifting) One-nat-def diff-zero nellist.map-cong0 split-beta)
 show ?thesis
 by (simp add: 6 8)
qed

lemma pifilt-upd-var:
assumes  ∃ x ∈ nset (ndropns σ). P ( x)

         ll = nmap (λ(x, k). if (P∘fst) x then nnth l (k − 1) else (snd x)) (cnt-in-nellist (nzip (ndropns σ)
l1) (P∘fst) 0)
        nlength (pifilt σ P) = nlength l
        ⋀i. i ≤ nlength σ ⟹
            P (ndropn i σ) = P (ndropn i (upd σ n ll ))
        nlength σ = nlength l1
shows  upd (pifilt σ P) n l = pifilt (upd σ n ll ) P
proof −
 have 1: nlength σ = nlength ll
   using assms   by (simp add: cnt-in-nellist-nlength ndropns-nlength)
 have 2: upd (pifilt σ P) n l =
        nmap (λ(x, y). if y then finsert n x else x |−| {|n|})
            (nzip (nmap (λs. nnth s 0) (nfilter P (ndropns σ))) l)
    unfolding upd-def pifilt-def sfxfilt-def by simp
 have 3: pifilt (upd σ n ll ) P =
        nmap (λs. nnth s 0)
            (nfilter P (ndropns (upd σ n ll)))
   unfolding  pifilt-def sfxfilt-def  by simp
 have 4: (ndropns (upd σ n ll)) =
        nmap (λ(s, ll). upd s n ll) (nzip (ndropns σ) (ndropns ll))
   using upd-ndropns    using 1 by auto
```

have *5*: (*nfilter P* (*nmap* (λ(*s*, *ll*). *upd s n ll*) (*nzip* (*ndropns σ*) (*ndropns ll*)))) =
nmap (λ(*s*, *ll*). *upd s n ll*)(*nfilter* (*P*∘(λ(*s*, *ll*). *upd s n ll*)) (*nzip* (*ndropns σ*) (*ndropns ll*)))
by (*simp add*: *nfilter-nmap*)
have *6*: *pifilt* (*upd σ n ll*) *P* =
upd (*pifilt σ P*) *n* (*nmap* (λ*s*. *nnth s 0*) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns σ*) (*ndropns ll*)))))
using *upd-pifilt*[*of σ ll P n*]
using *1 assms*(*4*) **by** *auto*
have *7*: *nlength l* = *nlength* (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns σ*) (*ndropns ll*))))
using *assms*(*3*) **unfolding** *pifilt-def sfxfilt-def*
by (*metis 1 ndropns-nlength nlength-nfilter-nmap nlength-nmap nmap-fst-nzip-a*)
have *8*: *l* = (*nmap* (λ*s*. *nnth s 0*) (*nmap snd* (*nfilter* (*P* ∘ *fst*) (*nzip* (*ndropns σ*) (*ndropns ll*)))))
using *cnt-in-nellist-nfilter-ndropns-var*[*of σ P ll l 0 l1*]

assms 7 **apply** *simp*
by (*metis* (*no-types*, *lifting*) *One-nat-def diff-zero nellist.map-cong0 split-beta*)
show *?thesis*
by (*simp add*: *6 8*)
qed

lemma *similar-upd*:
assumes *nlength σ* = *nlength l*
shows *similar σ n* (*upd σ n l*)
using *assms*
using *nlength-upd nnth-upd* **by** *auto*

lemma *exist-sigma-exist-value*:
(∃ *σ′*. *P σ′* ∧ (*similar σ n σ′*)) ⟷ (∃ *l*. *nlength σ* = *nlength l* ∧ *P* (*upd σ n l*))
proof −
have *2*: (∃*σ′*. *P σ′* ∧ (*similar σ n σ′*)) ⟹
(∃ *l*. *nlength σ* = *nlength l* ∧ *P* (*upd σ n l*))
proof −
assume *a0*: (∃*σ′*. *P σ′* ∧ (*similar σ n σ′*))
show (∃ *l*. *nlength σ* = *nlength l* ∧ *P* (*upd σ n l*))
proof −
obtain *σ′* **where** *3*: *P σ′* ∧ (*similar σ n σ′*)
using *a0* **by** *blast*
have *30*: *nlength σ′* = *nlength* (*upd σ′ n* (*nmap* (λ*x*. *n* |∈| *x*) *σ′*))
by (*simp add*: *nlength-upd*)
have *31*: ⋀*k* . *k* ≤ *nlength σ′* ⟹ *nnth σ′ k* = (*nnth* (*upd σ′ n* (*nmap* (λ*x*. *n* |∈| *x*) *σ′*)) *k*)
by (*simp add*: *finsert-absorb nnth-upd*)
have *4*: *σ′* = (*upd σ′ n* (*nmap* (λ*x*. *n* |∈| *x*) *σ′*))
by (*simp add*: *30 31 nellist-eq-nnth-eq*)
have *40*: *nlength σ* = *nlength* (*upd σ n* (*nmap* (λ*x*. *n* |∈| *x*) *σ*))
by (*simp add*: *nlength-upd*)
have *41*: ⋀*k*. *k*≤ *nlength σ* ⟹ *nnth σ k* = (*nnth* (*upd σ n* (*nmap* (λ*x*. *n* |∈| *x*) *σ*)) *k*)
by (*simp add*: *finsert-absorb nnth-upd*)
have *8*: *σ* = (*upd σ n* (*nmap* (λ*x*. *n* |∈| *x*) *σ*))

```
      by (simp add: 40 41 nellist-eq-nnth-eq)
    have 12: (similar σ n σ')
      using 3 by auto
    have 13: nlength σ = nlength σ'
      using 12 by auto
    have 14: ⋀ k . k ≤ nlength σ ⟹ (nnth σ' k) = (nnth (upd σ n (nmap (λx. n |∈| x) σ')) k)
      using 13 nnth-upd 12 by auto
    have 15: nlength σ' = nlength (upd σ n (nmap (λx. n |∈| x) σ'))
      by (simp add: 13 nlength-upd)
    have 16: σ' = (upd σ n (nmap (λx. n |∈| x) σ'))
      using nellist-eq-nnth-eq[of σ' (upd σ n (nmap (λx. n |∈| x) σ'))]
     using 13 14 15 by presburger
    show ?thesis
     by (metis 13 16 3 nlength-nmap)
    qed
   qed
  have 17: (∃ l. nlength σ = nlength l ∧ P (upd σ n l)  ) ⟹
          (∃σ'. P σ' ∧ (similar σ n σ'))
   proof −
    assume a1: (∃ l. nlength σ = nlength l ∧ P (upd σ n l)  )
    show (∃σ'. P σ'  ∧ (similar σ n σ'))
    proof −
     obtain l where 18:  nlength σ = nlength l ∧ P (upd σ n l)
       using a1 by blast
     have 19: similar σ n (upd σ n l)
       using 18 nlength-upd nnth-upd by auto
     show ?thesis
     using 18 19 by blast
    qed
   qed
  show ?thesis
  using 17 2 by blast
qed

lemma upd-self:
 shows  upd nell n (nmap (λ s. n |∈| s) nell) = nell
proof −
 have 1: nlength nell = nlength (upd nell n (nmap (λ s. n |∈| s) nell))
     by (simp add: nlength-upd)
 have 2: ⋀j. j ≤ nlength nell ⟹
          (nnth nell j) = (nnth (upd nell n (nmap (λ s. n |∈| s) nell)) j)
   using nnth-upd[of nell (nmap (λ s. n |∈| s) nell) - n]
    by auto
 show ?thesis
 by (simp add: 1 2 nellist-eq-nnth-eq)
qed
```

lemma *upd-undo*:
assumes *nlength l = nlength nell*
shows *upd (upd nell n l) n (nmap (λs. n |∈| s) nell) = nell*
using *assms*
using *upd-absorb upd-self* **by** *force*

lemma *nrev-upd*:
assumes *nfinite g*
nlength g = nlength l
shows *nrev (upd g n l) = (upd (nrev g) n (nrev l))*
proof −
have *1*: *nlength (nrev (upd g n l)) = nlength (upd (nrev g) n (nrev l))*
by (*simp add*: *assms*(*2*) *nlength-upd*)
have *2*: *⋀i. i ≤ nlength (nrev (upd g n l)) ⟶*
(nnth (nrev (upd g n l)) i) =
(nnth (upd (nrev g) n (nrev l)) i)
using *assms* **by** (*auto simp add*: *nlength-upd nnth-nrev nnth-upd nfinite-conv-nlength-enat*)
show *?thesis*
using *1 2 nellist-eq-nnth-eq* **by** *blast*
qed

lemma *upd-exist*:
∃ r2 ur2. nlength r2 = nlength r1 ∧ nlength ur2 = nlength r1 ∧ r1 = upd r2 n ur2
by (*metis nlength-nmap upd-self*)

lemma *pifilt-upd-undo*:
assumes *nlength σ = nlength l*
∃ x ∈ nset (ndropns s′). P x
upd σ n l = pifilt s′ P
ss′ = upd s′ n ll
ll= nmap (λ(x, k). if P x then nnth (nmap ((|∈|) n) σ) (k − 1) else True) (cnt-in-nellist (ndropns s′) P 0)
(⋀i. enat i ≤ nlength s′ ⟹ P (ndropn i s′) = P (ndropn i (upd s′ n ll)))
shows *σ = pifilt ss′ P*
proof −
have *1*: *upd (upd σ n l) n (nmap (λs. n |∈| s) σ) = σ*
using *assms* **by** (*metis upd-undo*)
have *2*: *nlength (pifilt s′ P) = nlength (nmap ((|∈|) n) σ)*
by (*metis assms*(*1,3*) *nlength-nmap nlength-upd*)
have *3*: *upd (pifilt s′ P) n (nmap (λs. n |∈| s) σ) = pifilt (upd s′ n ll) P*
using *assms pifilt-upd*[*of s′ P ll (nmap (λs. n |∈| s) σ) n*]
using *2* **by** *fastforce*
show *?thesis*
using *1 3 assms*(*3,4*) **by** *auto*
qed

lemma *pifilt-upd-undo-var*:
assumes *nlength σ = nlength l*
∃ x ∈ nset (ndropns s′). P x

upd σ *n l* = *pifilt s' P*
ss' = *upd s' n ll*
ll= *nmap* (λ(*x*, *k*). *if* (*P*∘*fst*) *x then nnth* (*nmap* ((|∈|) *n*) σ) (*k* − *1*) *else* (*snd x*)) (*cnt-in-nellist* (*nzip* (*ndropns s'*) *l1*) (*P*∘*fst*) *0*)
(⋀*i*. *enat i* ≤ *nlength s'* ⟹ *P* (*ndropn i s'*) = *P* (*ndropn i* (*upd s' n ll*)))
nlength s' = *nlength l1*
shows σ = *pifilt ss' P*
proof −
have *1*: *upd* (*upd* σ *n l*) *n* (*nmap* (λ*s*. *n* |∈| *s*) σ) = σ
using *assms* **by** (*metis upd-undo*)
have *2*: *nlength* (*pifilt s' P*) = *nlength* (*nmap* ((|∈|) *n*) σ)
by (*metis assms*(*1*,*3*) *nlength-nmap nlength-upd*)
have *3*: *upd* (*pifilt s' P*) *n* (*nmap* (λ*s*. *n* |∈| *s*) σ) = *pifilt* (*upd s' n ll*) *P*
using *assms pifilt-upd-var*[*of s' P ll* (*nmap* (λ*s*. *n* |∈| *s*) σ) *l1 n*]
using *2* **by** *blast*
show *?thesis*
using *1 3 assms*(*3*,*4*) **by** *auto*
qed

lemma *pfilt-upd-finite*:
assumes *nlength l* = *nlength* σ
nidx ls
nfinite σ
nfinite ls
(*enat* (*nlast ls*)) = (*nlength* σ)
shows *pfilt* (*upd* σ *n l*) *ls* = *upd* (*pfilt* σ *ls*) *n* (*nmap* (λ*x*. *nnth l x*) *ls*)
proof−
have *1*: *nlength* (*pfilt* (*upd* σ *n l*) *ls*) = *nlength* (*upd* (*pfilt* σ *ls*) *n* (*nmap* (λ*x*. *nnth l x*) *ls*))
using *assms pfilt-nmap* **by** (*simp add*: *nlength-upd*)
have *2*: ⋀*j*. *j*≤ *nlength* (*pfilt* (*upd* σ *n l*) *ls*) ⟹
nnth (*pfilt* (*upd* σ *n l*) *ls*) *j* =
nnth (*upd* (*pfilt* σ *ls*) *n* (*nmap* (λ*x*. *nnth l x*) *ls*)) *j*
using *assms pfilt-nmap*
by (*metis* (*full-types*) *enat-ord-simps*(*1*) *nidx-all-le-nlast nlength-nmap nnth-nmap nnth-upd*)
show *?thesis*
by (*metis 1 2 pfilt-expand*)
qed

lemma *pfilt-upd-infinite*:
assumes *nlength l* = *nlength* σ
nidx ls
¬*nfinite* σ
¬*nfinite ls*

shows *pfilt* (*upd* σ *n l*) *ls* = *upd* (*pfilt* σ *ls*) *n* (*nmap* (λ*x*. *nnth l x*) *ls*)
proof−
have *1*: *nlength* (*pfilt* (*upd* σ *n l*) *ls*) = *nlength* (*upd* (*pfilt* σ *ls*) *n* (*nmap* (λ*x*. *nnth l x*) *ls*))
using *assms pfilt-nmap* **by** (*simp add*: *nlength-upd*)

```
 have 2: ⋀j. j≤ nlength (pfilt (upd σ n l)  ls) ⟹
            nnth (pfilt (upd σ n l)  ls) j =
            nnth (upd (pfilt σ ls)  n (nmap (λx. nnth l x) ls)) j
       using assms pfilt-nmap        by (simp add: nfinite-conv-nlength-enat nnth-upd)
 show ?thesis
   by (metis 1 2 pfilt-expand)
qed

lemma upd-pfilt-finite:
 assumes nlength l = nlength (pfilt σ ls)
         nidx ls
         nfinite (pfilt σ ls)
         nfinite ls
          (enat (nlast ls)) = (nlength σ)
         nnth ls 0 = 0
         ll = nmap (λ (s,j). (if j ∈ nset ls then nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = j) else True)
) (nzip σ (niterates Suc 0))
 shows upd (pfilt σ ls) n l  = pfilt (upd σ n ll) ls
proof −
 have 1: nlength (upd (pfilt σ ls) n l) = nlength (pfilt (upd σ n ll) ls)
   using assms
   by (metis nlength-upd pfilt-expand)
 have 2: ⋀k. k≤ nlength (upd (pfilt σ ls) n l) ⟹
            nnth (upd (pfilt σ ls) n l) k =
            nnth (pfilt (upd σ n ll) ls) k
   proof −
    fix k :: nat
    assume a: enat k ≤ nlength (upd (pfilt σ ls) n l)
    show nnth (upd (pfilt σ ls) n l) k = nnth (pfilt (upd σ n ll) ls) k
    proof −
     have 3: (pfilt σ ls) = nmap (λ s. (nnth σ s)) ls
       using assms pfilt-nmap by auto
     have 4: nnth (upd (pfilt σ ls) n l) k =
               (if nnth l k then finsert n (nnth (pfilt σ ls) k) else (nnth (pfilt σ ls) k) |−| {|n|})
         using assms a            by (simp add: nlength-upd nnth-upd)
     have 5: (nnth (pfilt σ ls) k) = nnth σ (nnth ls k)
       using pfilt-nnth a assms nlength-upd by auto
     have 6: nnth (pfilt (upd σ n ll) ls) k = nnth (upd σ n ll) (nnth ls k)
        using 1 a pfilt-nnth by auto
     have 7: nlength σ = nlength ll
       using assms          by simp
     have 70: k ≤ nlength ls
        using 1 a pfilt-nlength by auto
     have 71: ⋀i. i ≤ nlength ls ⟹ (nnth ls i) ≤ nlength σ
       using assms nidx-all-le-nlast[of ls]          by (metis enat-ord-simps(1))
     have 8: enat (nnth ls k) ≤ nlength σ
      using a assms   using 70 71 by blast
     have 9: nnth (upd σ n ll) (nnth ls k) =
               (if nnth ll (nnth ls k) then finsert n (nnth σ (nnth ls k)) else (nnth σ (nnth ls k)) |−| {|n|})
      using assms nnth-upd[of σ ll (nnth ls k)]  7 8 by blast
```

```
    have 10:  enat (nnth ls k) ≤ nlength (nzip σ (niterates Suc 0))
      by (simp add: 8)
    have 11: nnth (nzip σ (niterates Suc 0)) (nnth ls k) = (nnth σ (nnth ls k), (nnth ls k))
      by (simp add: 8 nnth-nzip)
    have 12: nnth (nmap (λ (s,j). (if j ∈ nset ls then nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = j)
else True) ) (nzip σ (niterates Suc 0))) (nnth ls k) =
       (λ (s,j). (if j ∈ nset ls then nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = j) else True) )
         (nnth σ (nnth ls k), (nnth ls k))
       using assms nnth-nmap[of (nnth ls k) (nzip σ (niterates Suc 0))
         (λ (s,j). (if j ∈ nset ls then nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = j) else True) )] 10
      using 11 by presburger
    have 13: (λ (s,j). (if j ∈ nset ls then nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = j) else True) )
          (nnth σ (nnth ls k), (nnth ls k)) =
           (if (nnth ls k) ∈ nset ls then nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = (nnth ls k)) else
True)
        by simp
    have 14: (nnth ls k) ∈ nset ls
      by (metis a assms(1) in-nset-conv-nnth nlength-upd  pfilt-nlength)
    have 15: (THE y. y ≤ nlength ls  ∧ (nnth ls y) = (nnth ls k)) = k
     using 1 a assms(2) in-nset-conv-nnth[of - ls] ndistinct-Ex1[of ls] nidx-imp-ndistinct[of ls] pfilt-nlength[of
σ ls] the-sym-eq-trivial
        the1-equality[of λy. y ≤ nlength ls  ∧ (nnth ls y) = (nnth ls k) k]
        using assms(1) nlength-upd by auto
    have 16: nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = (nnth ls k)) = nnth l k
      using 15 by auto
    show ?thesis
    using 12 14 15 4 5 6 9 assms(7)  by force
   qed
  qed
 show ?thesis
 by (simp add: 1 2 nellist-eq-nnth-eq)
qed

lemma upd-pfilt-infinite:
 assumes nlength l = nlength (pfilt σ ls)
       nidx ls
       ¬nfinite (pfilt σ ls)
       ¬nfinite ls
       ¬nfinite σ
       nnth ls 0 = 0
       ll = nmap (λ (s,j). (if j ∈ nset ls then nnth l (THE y. y ≤ nlength ls  ∧ (nnth ls y) = j) else True)
) (nzip σ (niterates Suc 0))
 shows upd (pfilt σ ls) n l  = pfilt (upd σ n ll) ls
proof −
 have 1: nlength (upd (pfilt σ ls) n l) = nlength (pfilt (upd σ n ll) ls)
   using assms
   by (metis nlength-upd pfilt-expand)
 have 2: ⋀k. k≤ nlength (upd (pfilt σ ls) n l) ⟹
           nnth (upd (pfilt σ ls) n l) k =
           nnth (pfilt (upd σ n ll) ls) k
```

proof −
fix k :: *nat*
assume *a*: *enat* $k \leq$ *nlength* (*upd* (*pfilt* σ *ls*) *n l*)
show *nnth* (*upd* (*pfilt* σ *ls*) *n l*) k = *nnth* (*pfilt* (*upd* σ *n ll*) *ls*) k
proof −
have *3*: (*pfilt* σ *ls*) = *nmap* (λ *s*. (*nnth* σ *s*)) *ls*
using *assms pfilt-nmap* **by** *auto*
have *4*: *nnth* (*upd* (*pfilt* σ *ls*) *n l*) k =
(*if nnth l k then finsert n* (*nnth* (*pfilt* σ *ls*) k) *else* (*nnth* (*pfilt* σ *ls*) k) |−| {|*n*|})
using *assms a* **by** (*simp add*: *nlength-upd nnth-upd*)
have *5*: (*nnth* (*pfilt* σ *ls*) k) = *nnth* σ (*nnth ls k*)
using *pfilt-nnth a assms nlength-upd* **by** *auto*
have *6*: *nnth* (*pfilt* (*upd* σ *n ll*) *ls*) k = *nnth* (*upd* σ *n ll*) (*nnth ls k*)
using *1 a pfilt-nnth* **by** *auto*
have *7*: *nlength* σ = *nlength ll*
using *assms* **by** *simp*
have *70*: $k \leq$ *nlength ls*
using *1 a pfilt-nlength* **by** *auto*
have *71*: $\bigwedge i$. $i \leq$ *nlength ls* $\Longrightarrow$ (*nnth ls i*) $\leq$ *nlength* σ
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *8*: *enat* (*nnth ls k*) $\leq$ *nlength* σ
using *a assms* **using** *70 71* **by** *blast*
have *9*: *nnth* (*upd* σ *n ll*) (*nnth ls k*) =
(*if nnth ll* (*nnth ls k*) *then finsert n* (*nnth* σ (*nnth ls k*)) *else* (*nnth* σ (*nnth ls k*)) |−| {|*n*|})
using *assms nnth-upd*[*of* σ *ll* (*nnth ls k*)] *7 8* **by** *blast*
have *10*: *enat* (*nnth ls k*) $\leq$ *nlength* (*nzip* σ (*niterates Suc 0*))
by (*simp add*: *8*)
have *11*: *nnth* (*nzip* σ (*niterates Suc 0*)) (*nnth ls k*) = (*nnth* σ (*nnth ls k*), (*nnth ls k*))
by (*simp add*: *8 nnth-nzip*)
have *12*: *nnth* (*nmap* (λ (*s*,*j*). (*if j* $\in$ *nset ls then nnth l* (*THE y*. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = *j*) *else True*)) (*nzip* σ (*niterates Suc 0*))) (*nnth ls k*) =
(λ (*s*,*j*). (*if j* $\in$ *nset ls then nnth l* (*THE y*. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = *j*) *else True*))
(*nnth* σ (*nnth ls k*), (*nnth ls k*))
using *assms nnth-nmap*[*of* (*nnth ls k*) (*nzip* σ (*niterates Suc 0*))
(λ (*s*,*j*). (*if j* $\in$ *nset ls then nnth l* (*THE y*. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = *j*) *else True*))] *10*
using *11* **by** *presburger*
have *13*: (λ (*s*,*j*). (*if j* $\in$ *nset ls then nnth l* (*THE y*. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = *j*) *else True*))
(*nnth* σ (*nnth ls k*), (*nnth ls k*)) =
(*if* (*nnth ls k*) $\in$ *nset ls then nnth l* (*THE y*. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = (*nnth ls k*)) *else True*)
by *simp*
have *14*: (*nnth ls k*) $\in$ *nset ls*
by (*metis a assms*(*1*) *in-nset-conv-nnth nlength-upd pfilt-nlength*)
have *15*: (*THE y*. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = (*nnth ls k*)) = k
using *1 a assms*(*2*) *in-nset-conv-nnth*[*of - ls*] *ndistinct-Ex1*[*of ls*] *nidx-imp-ndistinct*[*of ls*] *pfilt-nlength*[*of* σ *ls*] *the-sym-eq-trivial*
the1-equality[*of* λy. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = (*nnth ls k*) k]
using *assms*(*1*) *nlength-upd* **by** *auto*
have *16*: *nnth l* (*THE y*. $y \leq$ *nlength ls* $\wedge$ (*nnth ls y*) = (*nnth ls k*)) = *nnth l k*

```
      using 15 by auto
    show ?thesis
    using 12 14 15 4 5 6 9 assms(7)  by force
   qed
  qed
 show ?thesis
 by (simp add: 1 2 nellist-eq-nnth-eq)
qed



lemma upd-pfilt-undo-finite:
assumes nlength σ = nlength l
        nidx ls
          nnth ls 0 = 0
         nfinite ls
         nfinite σ''
         nlast ls = the-enat (nlength σ'')
        upd σ n l = pfilt σ'' ls
        l1 = nmap (λ(s, j). if j ∈ nset ls then nnth (nmap ((|∈|) n) σ) (THE y. enat y ≤ nlength ls ∧ nnth
ls y = j) else True) (nzip σ'' (niterates Suc 0))
        σ' = (upd σ'' n l1)
shows  σ = pfilt σ' ls
proof −
 have 1:  upd (upd σ n l) n (nmap (λs. n |∈| s) σ) = σ
    using assms      by (metis upd-undo)
 have 2: nlength (nmap ((|∈|) n) σ) = nlength (pfilt σ'' ls)
    using assms      by (metis nlength-nmap nlength-upd)
 have 3:    upd (pfilt σ'' ls) n (nmap (λs. n |∈| s) σ) = pfilt (upd σ'' n l1) ls
    using assms 2 upd-pfilt-finite[of (nmap (λs. n |∈| s) σ) σ'' ls l1 n ]
      by (simp add: enat-the-enat nfinite-conv-nlength-enat pfilt-nlength)
 show ?thesis
 using 1 3 assms by auto
qed

lemma upd-pfilt-undo-infinite:
assumes nlength σ = nlength l
        nidx ls
          nnth ls 0 = 0
         ¬nfinite ls
         ¬nfinite σ''
        upd σ n l = pfilt σ'' ls
        l1 = nmap (λ(s, j). if j ∈ nset ls then nnth (nmap ((|∈|) n) σ) (THE y. enat y ≤ nlength ls ∧ nnth
ls y = j) else True) (nzip σ'' (niterates Suc 0))
        σ' = (upd σ'' n l1)
shows  σ = pfilt σ' ls
proof −
 have 1:  upd (upd σ n l) n (nmap (λs. n |∈| s) σ) = σ
    using assms      by (metis upd-undo)
 have 2: nlength (nmap ((|∈|) n) σ) = nlength (pfilt σ'' ls)
```

using *assms* **by** (*metis nlength-nmap nlength-upd*)
have *3*: *upd* (*pfilt* σ'' *ls*) *n* (*nmap* (λ*s*. *n* |∈| *s*) σ) = *pfilt* (*upd* σ'' *n l1*) *ls*
using *assms 2 upd-pfilt-infinite*[*of* (*nmap* (λ*s*. *n* |∈| *s*) σ) σ'' *ls l1 n*]
by (*simp add*: *enat-the-enat nfinite-conv-nlength-enat pfilt-nlength*)
show *?thesis*
using *1 3 assms* **by** *auto*
qed

lemma *upd-always-eqv*:
assumes (⋀*j*. *j*≤ *nlength s'* ⟹ (*n* |∈| (*nnth s' j*)) ⟷ *P* (*nnth s' j*))
shows (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*)) = *s'*
proof −
have *6030*: *nlength s'* = *nlength* (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*))
by (*simp add*: *nlength-upd*)
have *6040*: ⋀*j*. *j* ≤ *nlength s'* ⟹
(*nnth s' j*) = *nnth* (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*)) *j*
proof −
fix *j* :: *nat*
assume *bb*: *j* ≤ *nlength s'*
show (*nnth s' j*) = *nnth* (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*)) *j*
proof −
have *6050*: *nnth* (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*)) *j* =
(*if nnth* (*nmap* (λ*s*. *P s*) *s'*) *j then finsert n* (*nnth s' j*) *else nnth s' j* |−| {|*n*|})
using *nnth-upd*[*of s'* (*nmap* (λ*s*. *P s*) *s'*) *j n*] **by** (*simp add*: *bb*)
have *6060*: *nnth* (*nmap* (λ*s*. *P s*) *s'*) *j* = *P* (*nnth s' j*)
by (*simp add*: *bb*)
have *6070*: *P* (*nnth s' j*) ⟷ (*n* |∈| (*nnth s' j*))
using *bb assms* **by** *simp*
have *6080*: (*nnth s' j*) = (*if* (*n* |∈| (*nnth s' j*)) *then finsert n* (*nnth s' j*) *else nnth s' j* |−| {|*n*|})
by (*simp add*: *finsert-absorb*)
show *?thesis* **using** *6050 6060 6070 6080* **by** *metis*
qed
qed
have *6090*: (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*)) = *s'*
by (*simp add*: *6030 6040 nellist-eq-nnth-eq*)
show *?thesis* **using** *6090* **by** *metis*
qed

lemma *upd-always-eqv-var-1*:
assumes (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*)) = *s'*
shows (∀*j*. *j*≤ *nlength s'* ⟶ (*n* |∈| (*nnth s' j*)) ⟷ (*P* (*nnth s' j*)))
proof −
have *1*: *nlength s'* = *nlength* (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*))
using *assms* **by** *simp*
have *2*: ⋀*j*. *j* ≤ *nlength s'* ⟹
(*nnth s' j*) = *nnth* (*upd s' n* (*nmap* (λ*s*. *P s*) *s'*)) *j*
using *assms* **by** *auto*
show *?thesis*
using *2* **unfolding** *upd-def*

by (*metis* (*no-types*, *lifting*) *assms finsertCI fminus-iff nlength-nmap nnth-nmap nnth-upd*)
qed

lemma *upd-always-eqv-var-2*:
shows $((upd\ s'\ n\ (nmap\ (\lambda s.\ P\ s)\ s')) = s') \longleftrightarrow$
$(\forall j.\ j \leq nlength\ s' \longrightarrow (n\ |\in|\ (nnth\ s'\ j)) \longleftrightarrow (P\ (nnth\ s'\ j)\ \ ))$
by (*metis upd-always-eqv upd-always-eqv-var-1*)

lemma *upd-always-eqv-var-3-help*:
assumes $j \leq nlength\ s'$
shows $(nnth\ s'\ j = nnth\ (upd\ s'\ n\ (nmap\ (\lambda s.\ P\ s)\ (ndropns\ s'))))\ j) \longleftrightarrow$
$((n\ |\in|\ (nnth\ s'\ j)) \longleftrightarrow (P\ (ndropn\ j\ s')\ )\ )$
using *assms nnth-upd*[*of* s' $(nmap\ (\lambda s.\ P\ s)\ (ndropns\ s'))\ j\ n$]
by (*metis finsertCI finsert-absorb fminus-finsert-absorb fminus-idemp fminus-iff ndropns-nlength ndropns-nnth nlength-nmap nnth-nmap*)

lemma *upd-always-eqv-var-3*:
shows $((upd\ s'\ n\ (nmap\ (\lambda s.\ P\ s)\ (ndropns\ s'))) = s') \longleftrightarrow$
$(\forall j.\ j \leq nlength\ s' \longrightarrow (n\ |\in|\ (nnth\ s'\ j)) \longleftrightarrow (P\ (ndropn\ j\ s')\ )\ )$
using *upd-always-eqv-var-3-help*[*of* - s' n P] *nellist-eq-nnth-eq*[*of* $(upd\ s'\ n\ (nmap\ (\lambda s.\ P\ s)\ (ndropns\ s')))$ s']
by (*metis ndropns-nlength nlength-nmap nlength-upd*)

end

Chapter 3

Shallow embedding of Finite and Infinite ITL

3.1 Semantics

theory *Semantics*
imports *NELList-Extras HOL−TLA.Intensional*
begin

This theory mechanises a *shallow* embedding of Finite and Infinite ITL using the *NELList* and *Intensional* theories.

3.1.1 Types of Formulas

To mechanise the Finite and Infinite ITL semantics, the following type abbreviations are used:

type-synonym $'a$ *intervals* = $'a$ *nellist*

type-synonym ($'a$,$'b$) *formfun* = $'a$ *intervals* ⇒ $'b$
type-synonym $'a$ *formula* = ($'a$,*bool*) *formfun*
type-synonym ($'a$,$'b$) *stfun* = $'a$ ⇒ $'b$
type-synonym $'a$ *stpred* = ($'a$,*bool*) *stfun*

instance
fun :: (*type*,*type*) *world* **..**

instance
prod :: (*type*,*type*) *world* **..**

instance
sum :: (*type*,*type*) *world* **..**

instance
nellist :: (*type*) *world* **..**

Pair, function, sum, and interval are instantiated to be of type class world. This allows use of the lifted Intensional logic for formulas, and standard logical connectives can therefore be used.

3.1.2 Semantics of ITL

The semantics of ITL is defined.

definition *skip-d* :: (*'a* ::*world*) *formula*
where *skip-d* ≡ (*λs. nlength s* = (*enat* (*Suc 0*)))

definition *chop-d* :: (*'a* ::*world*) *formula* ⇒ (*'a* ::*world*) *formula* ⇒ (*'a* ::*world*) *formula*
where *chop-d F1 F2* ≡
(*λs.*
(∃ *n*≤*nlength s.* ((*ntaken n s*) ⊨ *F1*) ∧ ((*ndropn n s*) ⊨ *F2*))
∨ (¬ *nfinite s* ∧ (*s* ⊨ *F1*))
)

definition *current-val-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun*
where *current-val-d f* = (*λs.* ((*nfirst s*) ⊨ *f*))

definition *next-val-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun*
where *next-val-d f* ≡
(*λ s.* (*if nlength s* ≠ (*enat 0*) *then* ((*nnth s 1*) ⊨ *f*) *else* (ϵ (*x*::*'b*). *x*=*x*))
)

definition *fin-val-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun*
where *fin-val-d f* ≡ *λ s.* ((*if nfinite s then* ((*nlast s*) ⊨ *f*) *else* (ϵ (*x*::*'b*). *x*=*x*)))

definition *penult-val-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun*
where *penult-val-d f* ≡
(*λ s.*
(*if nfinite s*
then (*if nlength s* ≠ (*enat 0*)
then ((*nnth s* (*the-enat* (*epred*(*nlength s*)))) ⊨ *f*)
else (ϵ (*x*::*'b*). *x*=*x*))
else (ϵ (*x*::*'b*). *x*=*x*)
)
)

syntax
-skip-d :: *lift* ((*skip*))
-chop-d :: [*lift*,*lift*] ⇒ *lift* ((-;-) [*84*,*84*] *83*)
-current-val-d :: *lift* ⇒ *lift* (($-) [*100*] *99*)
-next-val-d :: *lift* ⇒ *lift* ((-$) [*100*] *99*)
-fin-val-d :: *lift* ⇒ *lift* ((!-) [*100*] *99*)
-penult-val-d :: *lift* ⇒ *lift* ((-!) [*100*] *99*)
TEMP :: *lift* ⇒ *'b* ((*TEMP* -))

syntax (*ASCII*)
-skip-d :: *lift* ((*skip*))
-chop-d :: [*lift*,*lift*] ⇒ *lift* ((-;-) [*84*,*84*] *83*)
-current-val-d :: *lift* ⇒ *lift* (($-) [*100*] *99*)
-next-val-d :: *lift* ⇒ *lift* ((-$) [*100*] *99*)

-fin-val-d :: *lift* ⇒ *lift* ((!-) [*100*] *99*)
-penult-val-d :: *lift* ⇒ *lift* ((-!) [*100*] *99*)

translations
-skip-d ⇌ *CONST skip-d*
-chop-d ⇌ *CONST chop-d*
-current-val-d ⇌ *CONST current-val-d*
-next-val-d ⇌ *CONST next-val-d*
-fin-val-d ⇌ *CONST fin-val-d*
-penult-val-d ⇌ *CONST penult-val-d*
TEMP F ⇀ (*F*:: (- *intervals*) ⇒ -)

3.1.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

definition *infinite-d* :: ('*a* ::*world*) *formula*
where *infinite-d* ≡ *LIFT*(#*True*;#*False*)

syntax
-infinite-d :: *lift* (*inf*)

syntax (*ASCII*)
-infinite-d :: *lift* (*inf*)

translations
-infinite-d ⇌ *CONST infinite-d*

definition *finite-d* :: ('*a* ::*world*) *formula*
where *finite-d* ≡ *LIFT*(¬(*inf*))

syntax
-finite-d :: *lift* (*finite*)

syntax (*ASCII*)
-finite-d :: *lift* (*finite*)

translations
-finite-d ⇌ *CONST finite-d*

definition *schop-d* :: ('*a*::*world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *schop-d F1 F2* ≡ *LIFT*((*F1* ∧ *finite*);*F2*)

definition *sometimes-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *sometimes-d F* ≡ *LIFT*(*finite*;*F*)

definition *di-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *di-d F* ≡ *LIFT*(*F*;#*True*)

definition *da-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*

where *da-d F* ≡ *LIFT*(*finite*;(*F*;#*True*))

definition *next-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *next-d F* ≡ *LIFT*(*skip*;*F*)

definition *prev-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *prev-d F* ≡ *LIFT*(*F*;*skip*)

syntax
-schop-d :: [*lift*,*lift*] ⇒ *lift* ((- ⌢ -) [*84*,*84*] *83*)
-sometimes-d :: *lift* ⇒ *lift* ((◇-) [*88*] *87*)
-di-d :: *lift* ⇒ *lift* ((*di* -) [*88*] *87*)
-da-d :: *lift* ⇒ *lift* ((*da* -) [*88*] *87*)
-next-d :: *lift* ⇒ *lift* ((○ -) [*88*] *87*)
-prev-d :: *lift* ⇒ *lift* ((*prev* -) [*88*] *87*)

syntax (*ASCII*)
-schop-d :: [*lift*,*lift*] ⇒ *lift* ((- *schop* -) [*84*,*84*] *83*)
-sometimes-d :: *lift* ⇒ *lift* ((<>-) [*88*] *87*)
-di-d :: *lift* ⇒ *lift* ((*di* -) [*88*] *87*)
-da-d :: *lift* ⇒ *lift* ((*da* -) [*88*] *87*)
-next-d :: *lift* ⇒ *lift* ((*next* -) [*88*] *87*)
-prev-d :: *lift* ⇒ *lift* ((*prev* -) [*88*] *87*)

translations
-schop-d ⇌ *CONST schop-d*
-sometimes-d ⇌ *CONST sometimes-d*
-di-d ⇌ *CONST di-d*
-da-d ⇌ *CONST da-d*
-next-d ⇌ *CONST next-d*
-prev-d ⇌ *CONST prev-d*

definition *df-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *df-d F* ≡ *LIFT*(*F*⌢#*True*)

definition *sda-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *sda-d F* ≡ *LIFT*(#*True*⌢(*F*⌢#*True*))

definition *always-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *always-d F* ≡ *LIFT*(¬(◇(¬*F*)))

definition *bi-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *bi-d F* ≡ *LIFT*(¬(*di*(¬*F*)))

definition *ba-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *ba-d F* ≡ *LIFT*(¬(*da*(¬*F*)))

definition *wnext-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*

where *wnext-d F* ≡ *LIFT*(¬(○(¬*F*)))

definition *wprev-d* :: (′*a*::*world*) *formula* ⇒ ′*a formula*
where *wprev-d F* ≡ *LIFT*(¬(*prev*(¬*F*)))

definition *more-d* :: (′*a*::*world*) *formula*
where *more-d* ≡ *LIFT*(○(#*True*))

syntax
-df-d :: *lift* ⇒ *lift* ((*df* -) [*88*] *87*)
-sda-d :: *lift* ⇒ *lift* ((*sda* -) [*88*] *87*)
-always-d :: *lift* ⇒ *lift* ((□ -) [*88*] *87*)
-bi-d :: *lift* ⇒ *lift* ((*bi* -) [*88*] *87*)
-ba-d :: *lift* ⇒ *lift* ((*ba* -) [*88*] *87*)
-wnext-d :: *lift* ⇒ *lift* ((*wnext* -) [*88*] *87*)
-wprev-d :: *lift* ⇒ *lift* ((*wprev* -) [*88*] *87*)
-more-d :: *lift* ((*more*))

syntax (*ASCII*)
-df-d :: *lift* ⇒ *lift* ((*df* -) [*88*] *87*)
-sda-d :: *lift* ⇒ *lift* ((*sda* -) [*88*] *87*)
-always-d :: *lift* ⇒ *lift* (([] -) [*88*] *87*)
-bi-d :: *lift* ⇒ *lift* ((*bi* -) [*88*] *87*)
-ba-d :: *lift* ⇒ *lift* ((*ba* -) [*88*] *87*)
-wnext-d :: *lift* ⇒ *lift* ((*wnext* -) [*88*] *87*)
-wprev-d :: *lift* ⇒ *lift* ((*wprev* -) [*88*] *87*)
-more-d :: *lift* ((*more*))

translations
-df-d ⇌ *CONST df-d*
-sda-d ⇌ *CONST sda-d*
-always-d ⇌ *CONST always-d*
-bi-d ⇌ *CONST bi-d*
-ba-d ⇌ *CONST ba-d*
-wnext-d ⇌ *CONST wnext-d*
-wprev-d ⇌ *CONST wprev-d*
-more-d ⇌ *CONST more-d*

definition *bf-d* :: (′*a*::*world*) *formula* ⇒ ′*a formula*
where *bf-d F* ≡ *LIFT*(¬(*df*(¬*F*)))

definition *sba-d* :: (′*a*::*world*) *formula* ⇒ ′*a formula*
where *sba-d F* ≡ *LIFT*(¬(*sda*(¬*F*)))

definition *empty-d* :: (′*a*::*world*) *formula*
where *empty-d* ≡ *LIFT*(¬(*more*))

definition *fmore-d* :: (*'a*::*world*) *formula*
where *fmore-d* ≡ *LIFT*(*more* ∧ *finite*)

definition *dm-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *dm-d F* ≡ *LIFT*(#*True*;(*more* ∧ *F*))

syntax
-bf-d :: *lift* ⇒ *lift* ((*bf* -) [*88*] *87*)
-sba-d :: *lift* ⇒ *lift* ((*sba* -) [*88*] *87*)
-empty-d :: *lift* ((*empty*))
-fmore-d :: *lift* ((*fmore*))
-dm-d :: *lift* ⇒ *lift* ((*dm* -) [*88*] *87*)

syntax (*ASCII*)
-bf-d :: *lift* ⇒ *lift* ((*bf* -) [*88*] *87*)
-sba-d :: *lift* ⇒ *lift* ((*sba* -) [*88*] *87*)
-empty-d :: *lift* ((*empty*))
-fmore-d :: *lift* ((*fmore*))
-dm-d :: *lift* ⇒ *lift* ((*dm* -) [*88*] *87*)

translations
-bf-d ⇌ *CONST bf-d*
-sba-d ⇌ *CONST sba-d*
-empty-d ⇌ *CONST empty-d*
-fmore-d ⇌ *CONST fmore-d*
-dm-d ⇌ *CONST dm-d*

definition *bm-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *bm-d F* ≡ *LIFT*(¬(*dm*(¬*F*)))

definition *init-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *init-d F* ≡ *LIFT*((*empty* ∧ *F*);#*True*)

definition *fin-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *fin-d F* ≡ *LIFT*(□(*empty* ⟶ *F*))

definition *halt-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *halt-d F* ≡ *LIFT*(□(*empty* = *F*))

definition *initonly-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *initonly-d F* ≡ *LIFT*(*bi*(*empty* = *F*))

definition *keep-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *keep-d F* ≡ *LIFT*(*ba*(*skip* ⟶ *F*))

definition *yields-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *yields-d F1 F2* ≡ *LIFT*(¬(*F1*;(¬*F2*)))

definition *syields-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *syields-d F1 F2* ≡ *LIFT*(¬(*F1*⌢(¬*F2*)))

definition *ifthenelse-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula* ⇒ *'a formula*
where *ifthenelse-d F G H* ≡ *LIFT*((*F* ∧ *G*) ∨ (¬*F* ∧ *H*))

definition *revyields-d* :: (*'a* ::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *revyields-d F1 F2* ≡ *LIFT*(¬((¬ *F1*);*F2*))

definition *revsyields-d* :: (*'a* ::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *revsyields-d F1 F2* ≡ *LIFT*(¬((¬ *F1*)⌢*F2*))

syntax

-bm-d	:: *lift* ⇒ *lift*	((*bm* -) [*88*] *87*)
-init-d	:: *lift* ⇒ *lift*	((*init* -) [*88*] *87*)
-fin-d	:: *lift* ⇒ *lift*	((*fin* -) [*88*] *87*)
-halt-d	:: *lift* ⇒ *lift*	((*halt* -) [*88*] *87*)
-initonly-d	:: *lift* ⇒ *lift*	((*initonly* -) [*88*] *87*)
-keep-d	:: *lift* ⇒ *lift*	((*keep* -) [*88*] *87*)
-yields-d	:: [*lift*,*lift*] ⇒ *lift*	((- |∼> -) [*88*,*88*] *87*)
-syields-d	:: [*lift*,*lift*] ⇒ *lift*	((- |≃> -) [*88*,*88*] *87*)
-ifthenelse-d	:: [*lift*,*lift*,*lift*] ⇒ *lift*	((if_i - *then* - *else* -) [*88*,*88*,*88*] *87*)
-revyields-d	:: [*lift*,*lift*] ⇒ *lift*	((- <∼| -) [*84*,*84*] *83*)
-revsyields-d	:: [*lift*,*lift*] ⇒ *lift*	((- <≃| -) [*84*,*84*] *83*)

syntax (*ASCII*)

-bm-d	:: *lift* ⇒ *lift*	((*bm* -) [*88*] *87*)
-init-d	:: *lift* ⇒ *lift*	((*init* -) [*88*] *87*)
-fin-d	:: *lift* ⇒ *lift*	((*fin* -) [*88*] *87*)
-halt-d	:: *lift* ⇒ *lift*	((*halt* -) [*88*] *87*)
-initonly-d	:: *lift* ⇒ *lift*	((*initonly* -) [*88*] *87*)
-keep-d	:: *lift* ⇒ *lift*	((*keep* -) [*88*] *87*)
-yields-d	:: [*lift*,*lift*] ⇒ *lift*	((- *yields* -) [*88*,*88*] *87*)
-syields-d	:: [*lift*,*lift*] ⇒ *lift*	((- *syields* -) [*88*,*88*] *87*)
-ifthenelse-d	:: [*lift*,*lift*,*lift*] ⇒ *lift*	((if_i - *then* - *else* -) [*88*,*88*,*88*] *87*)
-revyields-d	:: [*lift*,*lift*] ⇒ *lift*	((- *revyields* -) [*84*,*84*] *83*)
-revsyields-d	:: [*lift*,*lift*] ⇒ *lift*	((- *revsyields* -) [*84*,*84*] *83*)

translations

-bm-d	⇌ *CONST bm-d*
-init-d	⇌ *CONST init-d*
-fin-d	⇌ *CONST fin-d*
-halt-d	⇌ *CONST halt-d*
-initonly-d	⇌ *CONST initonly-d*
-keep-d	⇌ *CONST keep-d*
-yields-d	⇌ *CONST yields-d*
-syields-d	⇌ *CONST syields-d*

-ifthenelse-d $\rightleftharpoons$ *CONST ifthenelse-d*
-revyields-d $\rightleftharpoons$ *CONST revyields-d*
-revsyields-d $\rightleftharpoons$ *CONST revsyields-d*

definition *sfin-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula*
where *sfin-d F* $\equiv$ *LIFT*($\neg$ (*fin* ($\neg$ *F*)))

definition *ifthen-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *ifthen-d F G* $\equiv$ *LIFT*(*if*$_i$ *F then G else #True*)

syntax
-ifthen-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((*if*$_i$ - *then* -) [*88*,*88*] *87*)
-sfin-d :: *lift* $\Rightarrow$ *lift* ((*sfin* -) [*88*] *87*)

syntax (*ASCII*)
-ifthen-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((*if*$_i$ - *then* -) [*88*,*88*] *87*)

-sfin-d :: *lift* $\Rightarrow$ *lift* ((*sfin* -) [*88*] *87*)

translations
-ifthen-d $\rightleftharpoons$ *CONST ifthen-d*
-sfin-d $\rightleftharpoons$ *CONST sfin-d*

definition *afb-d* :: (*'a* ::*world*) *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *afb-d F1 F2* $\equiv$ *LIFT*(*bf*(*F1* $\longrightarrow$ *fin F2*))

definition *safb-d* :: (*'a* ::*world*) *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *safb-d F1 F2* $\equiv$ *LIFT*(*bf*(*F1* = *fin F2*))

syntax
-afb-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mapsto$ -) [*84*,*84*] *83*)
-safb-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\longleftrightarrow$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-afb-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *afb* -) [*84*,*84*] *83*)
-safb-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *safb* -) [*84*,*84*] *83*)

translations
-afb-d $\rightleftharpoons$ *CONST afb-d*
-safb-d $\rightleftharpoons$ *CONST safb-d*

definition *next-assign-d* :: (*'a*::*world*,*'b*) *stfun* $\Rightarrow$ (*'a*,*'b*) *formfun* $\Rightarrow$ *'a formula*
where *next-assign-d v e* $\equiv$ *LIFT*(*v*$ = *e*)

definition *prev-assign-d* :: (*'a*::*world*,*'b*) *stfun* $\Rightarrow$ (*'a*,*'b*) *formfun* $\Rightarrow$ *'a formula*
where *prev-assign-d v e* $\equiv$ *LIFT*(*finite* $\longrightarrow$ *v*! = *e*)

definition *always-eq-d* :: ('a::*world*,'b) *stfun* ⇒ ('a,'b) *formfun* ⇒ 'a *formula*
where *always-eq-d v e* ≡ λ *s*. *s* ⊨ □($*v* = *e*)

definition *temporal-assign-d* :: ('a::*world*,'b) *stfun* ⇒ ('a,'b) *formfun* ⇒ 'a *formula*
where *temporal-assign-d v e* ≡ λ *s*. *s* ⊨ *finite* ⟶ !*v* = *e*

definition *gets-d* :: ('a::*world*,'b) *stfun* ⇒ ('a,'b) *formfun* ⇒ 'a *formula*
where *gets-d v e* ≡ λ *s*. *s* ⊨ *keep*(*temporal-assign-d v e*)

definition *stable-d* :: ('a::*world*,'b) *stfun* ⇒ 'a *formula*
where *stable-d v* ≡ λ *s*. *s* ⊨ *gets-d v* (*current-val-d v*)

definition *padded-d* :: ('a::*world*,'b) *stfun* ⇒ 'a *formula*
where *padded-d v* ≡ λ *s*. *s* ⊨ (*stable-d v*);*skip* ∨ *empty*

definition *padded-temp-assign-d* :: ('a::*world*,'b) *stfun* ⇒ ('a,'b) *formfun* ⇒ 'a *formula*
where *padded-temp-assign-d v e* ≡ λ *s*. *s* ⊨ (*temporal-assign-d v e*) ∧ (*padded-d v*)

syntax
-next-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- := -) [*50*,*51*] *50*)
-prev-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- =: -) [*50*,*51*] *50*)
-always-eq-d :: [*lift*,*lift*] ⇒ *lift* ((- ≈ -) [*50*,*51*] *50*)
-temporal-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- ← -) [*50*,*51*] *50*)
-gets-d :: [*lift*,*lift*] ⇒ *lift* ((- *gets* -) [*50*,*51*] *50*)
-stable-d :: *lift* ⇒ *lift* ((*stable* -) [*51*] *50*)
-padded-d :: *lift* ⇒ *lift* ((*padded* -) [*51*] *50*)
-padded-temp-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- <∼ -) [*50*,*51*] *50*)

syntax (*ASCII*)
-next-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- := -) [*50*,*51*] *50*)
-prev-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- =: -) [*50*,*51*] *50*)
-always-eq-d :: [*lift*,*lift*] ⇒ *lift* ((- *alweqv* -) [*50*,*51*] *50*)
-temporal-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- <−− -) [*50*,*51*] *50*)
-gets-d :: [*lift*,*lift*] ⇒ *lift* ((- *gets* -) [*50*,*51*] *50*)
-stable-d :: *lift* ⇒ *lift* ((*stable* -) [*51*] *50*)
-padded-d :: *lift* ⇒ *lift* ((*padded* -) [*51*] *50*)
-padded-temp-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- <~ -) [*50*,*51*] *50*)

translations
-next-assign-d ⇌ *CONST next-assign-d*
-prev-assign-d ⇌ *CONST prev-assign-d*
-always-eq-d ⇌ *CONST always-eq-d*
-temporal-assign-d ⇌ *CONST temporal-assign-d*
-gets-d ⇌ *CONST gets-d*
-stable-d ⇌ *CONST stable-d*
-padded-d ⇌ *CONST padded-d*
-padded-temp-assign-d ⇌ *CONST padded-temp-assign-d*

lemmas *itl-def = skip-d-def chop-d-def current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def infinite-d-def finite-d-def schop-d-def sometimes-d-def di-d-def da-d-def next-d-def prev-d-def df-d-def sda-d-def always-d-def bi-d-def ba-d-def wnext-d-def wprev-d-def more-d-def bf-d-def sba-d-def empty-d-def fmore-d-def dm-d-def bm-d-def init-d-def fin-d-def halt-d-def initonly-d-def keep-d-def yields-d-def syields-d-def ifthenelse-d-def sfin-d-def ifthen-d-def next-assign-d-def prev-assign-d-def always-eq-d-def temporal-assign-d-def gets-d-def stable-d-def padded-d-def padded-temp-assign-d-def revyields-d-def revsyields-d-def afb-d-def safb-d-def*

3.1.4 Properties of Operators

The following lemmas show that above operators have the expected semantics.

lemma *skip-defs* :
$(w \models skip) = (nlength\ w = (enat\ 1))$
by (*simp add*: *itl-def*)

lemma *chop-defs* :
$(w \models F1\ ;\ F2) =$
(
$(\exists\, n{\leq}nlength\ w.\ (\ (ntaken\ n\ w) \models F1\ )\ \wedge\ ((ndropn\ n\ w) \models F2\ )\ )$
$\vee\ (\neg\ nfinite\ w \wedge (\ w \models F1))$
)
by (*simp add*: *itl-def*)

lemma *yields-defs* :
$(w \models F1\ yields\ F2) =$
$((\forall\ n.\ ((ntaken\ n\ w) \models F1)\ \longrightarrow enat\ n \leq nlength\ w \longrightarrow\ (ndropn\ n\ w \models\ F2)) \wedge (nfinite\ w \vee (w \models \neg F1)\ ))$

by (*simp add*: *itl-def*)

lemma *revyields-defs* :
$(w \models F1\ revyields\ F2) =$
$((\forall\ n.\ ((ndropn\ n\ w) \models F2)\ \longrightarrow enat\ n \leq nlength\ w \longrightarrow\ ((ntaken\ n\ w) \models\ F1)) \wedge (nfinite\ w \vee (w \models F1)\ ))$

by (*simp add*: *itl-def*)
blast

lemma *revsyields-defs* :
$(w \models F1\ revsyields\ F2) =$
$(\forall\ n.\ (\ (ndropn\ n\ w) \models F2)\ \longrightarrow enat\ n \leq nlength\ w \longrightarrow\ ((ntaken\ n\ w \models\ F1)))$

by (*simp add*: *itl-def*)
blast

lemma *infinite-defs*:
$(w \models inf) = (\neg\ nfinite\ w)$
by (*simp add*: *itl-def*)

lemma *finite-defs* :
$(w \models \mathit{finite}) = (\mathit{nfinite}\ w)$
by (*simp add*: *itl-def*)

lemma *schop-defs* :
$(w \models \mathit{F1} \frown \mathit{F2}) =$
(
$(\exists\, n{\leq}\mathit{nlength}\ w.\ (\ (\mathit{ntaken}\ n\ w) \models \mathit{F1}\ )\ \wedge (\ (\mathit{ndropn}\ n\ w) \models \mathit{F2}\ )\ )$
)

by (*auto simp add*: *itl-def chop-defs finite-defs*)

lemma *syields-defs* :
$(w \models \mathit{F1\ syields\ F2}) =$
$(\forall\ n.\ (\ (\mathit{ntaken}\ n\ w) \models \mathit{F1}) \longrightarrow \mathit{enat}\ n \leq \mathit{nlength}\ w \longrightarrow ((\mathit{ndropn}\ n\ w \models\ \mathit{F2})))$

by (*simp add*: *itl-def*)

lemma *sometimes-defs* :
$(w \models \Diamond\ F) = (\exists\ n{\leq}\mathit{nlength}\ w.\ (\ (\mathit{ndropn}\ n\ w)) \models F)$
by (*simp add*: *itl-def finite-defs chop-defs*)

lemma *always-defs* :
$(w \models \Box\ F) =$
$(\forall\ n{\leq}\mathit{nlength}\ w.\ (\ (\mathit{ndropn}\ n\ w) \models F))$
by (*simp add*: *itl-def sometimes-defs*)

lemma *di-defs* :
$(w \models \mathit{di}\ F) =$
$((\exists\ n{\leq}\mathit{nlength}\ w.\ ((\mathit{ntaken}\ n\ w) \models F)) \vee (\neg \mathit{nfinite}\ w \wedge (w \models F)))$
by (*simp add*: *itl-def*)

lemma *df-defs* :
$(w \models \mathit{df}\ F) =$
$(\exists\ n{\leq}\mathit{nlength}\ w.\ ((\mathit{ntaken}\ n\ w) \models F))$
by (*simp add*: *df-d-def schop-defs*)

lemma *bi-defs* :
$(w \models \mathit{bi}\ F) =$
$((\forall\ n{\leq}\mathit{nlength}\ w\ .\ ((\mathit{ntaken}\ n\ w) \models F)) \wedge(\mathit{nfinite}\ w \vee (w \models F)))$
by (*simp add*: *itl-def di-defs*)

lemma *bf-defs* :
$(w \models \mathit{bf}\ F) =$
$(\forall\ n{\leq}\mathit{nlength}\ w.\ (\ (\mathit{ntaken}\ n\ w) \models F))$
by (*simp add*: *bf-d-def df-defs*)

lemma *da-defs* :
$(w \models \mathit{da}\ F) =$

((∃ n na. (n+na ≤ nlength w ∧ ((nsubn w n (n+ na)) ⊨ F)) ∨ (¬nfinite w ∧ ((ndropn n w) ⊨ F)))
)
proof
(*auto simp add*: *itl-def chop-defs nsubn-def1*)
show ⋀*n na*.
enat n ≤ *nlength w* ⟹
enat na ≤ *nlength w* − *enat n* ⟹
F (*ntaken na* (*ndropn n w*)) ⟹ ∃ *n*. (∃ *na*. *enat* (*n* + *na*) ≤ *nlength w* ∧ *F* (*ntaken na* (*ndropn n w*)))
∨ ¬ *nfinite w* ∧ *F* (*ndropn n w*)
by (*metis add-left-mono enat.simps*(*3*) *enat-add-sub-same le-iff-add plus-enat-simps*(*1*))
next
fix *n* :: *nat* **and** *na* :: *nat*
assume *a1*: *enat* (*n* + *na*) ≤ *nlength w*
assume *a2*: *F* (*ntaken na* (*ndropn n w*))
have *enat n* ≤ *nlength w*
using *a1* **by** (*meson enat-ord-simps*(*1*) *le-iff-add order-subst2*)
then show ∃ *n*. *enat n* ≤ *nlength w* ∧
((∃ *na*. *enat na* ≤ *nlength w* − *enat n* ∧ *F* (*ntaken na* (*ndropn n w*)))
∨ ¬ *nfinite w* ∧ *F* (*ndropn n w*))
using *a2 a1* **by** (*metis* (*no-types*) *add-diff-cancel-left*′ *enat-minus-mono1 idiff-enat-enat*)
next
show ⋀*n*. ¬ *nfinite w* ⟹
F (*ndropn n w*) ⟹
∃ *n*. *enat n* ≤ *nlength w* ∧
((∃ *na*. *enat na* ≤ *nlength w* − *enat n* ∧ *F* (*ntaken na* (*ndropn n w*))) ∨ *F* (*ndropn n w*))
by (*meson enat-ile le-cases nfinite-conv-nlength-enat*)
qed

lemma *ba-defs* :
(*w* ⊨ *ba F*) =
(∀ *n na*. (*enat* (*n* + *na*) ≤ *nlength w* ⟶ ((*nsubn w n* (*n*+*na*)) ⊨ *F*))
∧ (*nfinite w* ∨ ((*ndropn n w*) ⊨ *F*)))
by (*simp add*: *ba-d-def da-defs*)

lemma *sda-defs* :
(*w* ⊨ *sda F*) =
(∃ *n na*. (*n*+*na* ≤ *nlength w* ∧ ((*nsubn w n* (*n*+ *na*)) ⊨ *F*)))
proof
(*auto simp add*: *sda-d-def schop-defs nsubn-def1*)
show ⋀*n na*.
enat n ≤ *nlength w* ⟹
enat na ≤ *nlength w* − *enat n* ⟹ *F* (*ntaken na* (*ndropn n w*)) ⟹
∃ *n na*. *enat* (*n* + *na*) ≤ *nlength w* ∧ *F* (*ntaken na* (*ndropn n w*))
by (*metis add-le-cancel-left enat-ord-simps*(*1*) *idiff-enat-enat le-add-diff-inverse le-cases*
nfinite-nlength-enat nfinite-ntaken ntaken-all)
next
fix *n* :: *nat* **and** *na* :: *nat*
assume *a1*: *F* (*ntaken na* (*ndropn n w*))
assume *a2*: *enat* (*n* + *na*) ≤ *nlength w*

have *enat na ≤ nlength w − enat n*
by (*metis a2 add-diff-cancel-left′ enat-minus-mono1 idiff-enat-enat*)
then show ∃ *n. enat n ≤ nlength w ∧*
(∃ *na. enat na ≤ nlength w − enat n ∧ F* (*ntaken na* (*ndropn n w*)))
using *a2 a1*
by (*metis add.right-neutral le-cases le-zero-eq ndropn-all ndropn-nlength nlength-NNil*
the-enat.simps the-enat-0)
qed

lemma *sba-defs* :
(*w* ⊨ *sba F*) =
(∀ *n na. n*+*na* ≤ *nlength w* ⟶ ((*nsubn w n* (*n*+*na*)) ⊨ *F*))
by (*simp add*: *sba-d-def sda-defs*)

lemma *next-defs* :
(*w* ⊨ ○ *F*) =
(*nlength w* ≠ (*enat 0*) ∧ ((*ndropn 1 w*)⊨ *F*))
by (*simp add*: *itl-def chop-defs*)
(*metis One-nat-def Suc-ile-eq dual-order.order-iff-strict enat-le-plus-same*(*1*) *gen-nlength-def*
min.orderE min-enat-simps(*2*) *nlength-code nlength-eq-enat-nfiniteD one-enat-def*
the-enat.simps zero-enat-def zero-one-enat-neq(*1*))

lemma *wnext-defs* :
(*w* ⊨ *wnext F*) =
(*nlength w* = (*enat 0*) ∨ ((*ndropn 1 w*)⊨ *F*))
by (*simp add*: *wnext-d-def next-defs*)

lemma *prev-defs* :
(*w* ⊨ *prev F*) =
((*nlength w* ≠ (*enat 0*) ∧ *nfinite w* ∧
((*ntaken* (*the-enat*((*epred*(*nlength w*)))) *w*) ⊨ *F*)) ∨ (¬*nfinite w* ∧ (*w* ⊨ *F*)))
proof (*cases nfinite w*)
case *True*
then show *?thesis*
by (*auto simp add*: *prev-d-def chop-defs skip-defs*)
(*metis One-nat-def diff-diff-cancel enat-ord-simps*(*1*) *epred-enat idiff-enat-enat nfinite-nlength-enat*
the-enat.simps,
metis One-nat-def diff-le-self eSuc-epred enat.simps(*3*) *enat-add-sub-same enat-ord-simps*(*1*)
epred-enat nfinite-nlength-enat one-enat-def plus-1-eSuc(*2*) *the-enat.simps zero-enat-def*)
next
case *False*
then show *?thesis*
by (*auto simp add*: *prev-d-def chop-defs skip-defs*)
(*metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD*)
qed

lemma *wprev-defs* :
(*w* ⊨ *wprev F*) =
((*nfinite w* ⟶ (*nlength w* = (*enat 0*) ∨ ((*ntaken* (*the-enat*(*epred*(*nlength w*))) *w*) ⊨ *F*)))
∧ (*nfinite w* ∨ (*w* ⊨ *F*)))

by (*simp add*: *wprev-d-def prev-defs*)

lemma *more-defs* :
(*w* ⊨ *more*) = (*0* < *nlength w*)
using *zero-enat-def* **by** (*auto simp add*: *more-d-def next-defs*)

lemma *fmore-defs* :
(*w* ⊨ *fmore*) = (*nfinite w* ∧ (*0* < *nlength w*))
by (*auto simp add*: *fmore-d-def more-defs finite-defs*)

lemma *empty-defs* :
(*w* ⊨ *empty*) = (*nlength w* = *0*)
by (*simp add*: *empty-d-def more-defs*)

lemma *init-defs* :
(*w* ⊨ *init F*) = ((*ntaken 0 w*) ⊨ *F*)
by (*simp add*: *init-d-def chop-defs empty-defs min-def*)
(*metis ntaken-0 ntaken-all the-enat.simps zero-enat-def zero-le*)

lemma *initalt-defs* :
(*w* ⊨ *bi*(*empty* ⟶ *F*)) = ((*ntaken 0 w*) ⊨ *F*)
by (*simp add*: *bi-defs empty-defs min-def*)
(*metis enat-0-iff*(*2*) *nlength-eq-enat-nfiniteD ntaken-0 zero-le*)

lemma *fin-defs* :
(*w* ⊨ *fin F*) =
((*nfinite w* ∧ ((*ndropn* (*the-enat*(*nlength w*)) *w*) ⊨ *F*)) ∨ (¬*nfinite w*))
by (*simp add*: *fin-d-def empty-defs always-defs*)
(*metis add.right-neutral enat.distinct*(*2*) *enat-add-sub-same le-iff-add nfinite-nlength-enat*
nlength-eq-enat-nfiniteD the-enat.simps)

lemma *finalt-defs* :
(*w* ⊨ #*True*;(*F* ∧ *empty*)) =
((*nfinite w* ∧ ((*ndropn* (*the-enat*(*nlength w*)) *w*) ⊨ *F*)) ∨ (¬*nfinite w*))
by (*simp add*: *chop-defs empty-defs*)
(*metis add.right-neutral enat.distinct*(*2*) *enat-add-sub-same le-iff-add nfinite-nlength-enat*
the-enat.simps)

lemma *sfin-defs* :
(*w* ⊨ *sfin F*) = (*nfinite w* ∧ ((*ndropn* (*the-enat*(*nlength w*)) *w*) ⊨ *F*))
by (*auto simp add*: *sfin-d-def fin-defs*)

lemma *afb-defs*:
(*w* ⊨ *f1 afb f2*) =
(∀ *n*. ((*ntaken n w*) ⊨ *f1*) ∧ *enat n* ≤ *nlength w* ⟶ ((*NNil* (*nnth w n*)) ⊨ *f2*))
unfolding *afb-d-def*
using *bf-defs*[*of LIFT*(*f1* ⟶ *fin f2*)] *fin-defs*[*of f2*]
by *simp*
(*metis min.absorb1 ndropn-all ntaken-nlast ntaken-nlength order-refl*)

lemma *safb-defs*:
$(w \models f1\ safb\ f2) =$
$(\forall\ n.\ \ enat\ n \leq nlength\ w\ \longrightarrow ((ntaken\ n\ w) \models f1) = ((NNil\ (nnth\ w\ n)) \models f2))$
unfolding *safb-d-def*
using *bf-defs*[*of LIFT*(*f1* = *fin f2*)] *fin-defs*[*of f2*]
by (*simp add*: *ndropn-all ntaken-nlast*)

lemma *halt-defs* :
$(w \models halt(F)) = (\forall\, n.\ n \leq nlength\ w \longrightarrow (nlength\ w = n) = (\ \ (ndropn\ n\ w) \models F))$

by (*simp add*: *halt-d-def empty-defs always-defs*)
(*metis add.right-neutral dual-order.strict-iff-order enat-add-sub-same enat-ord-code*(*4*)
le-iff-add)

lemma *initonly-defs* :
$(w \models initonly(F)) =$
$(\ (\forall\, n.\ n \leq nlength\ w \longrightarrow (n = 0) = (\ (ntaken\ \ n\ w) \models F\ ))\wedge$
$(nfinite\ w \vee (nlength\ w = 0) = F\ w)$
)
by (*simp add*: *initonly-d-def bi-defs empty-defs zero-enat-def*)

lemma *ifthenelse-defs*:
$(w \models if_i\ F\ then\ G\ else\ H) =$
$(\ ((w \models F) \wedge (w \models G)) \vee ((\ \neg(w \models F) \wedge (w \models H)))\ )$
by (*simp add*: *itl-def*)

lemma *currentval-defs* :
$(s \models \$v) = (v\ (nfirst\ s))$
by (*simp add*: *itl-def*)

lemma *nextval-defs* :
$(s \models v\$) =$
(*if nlength s* $\neq$ (*enat 0*) *then* (*v* (*nnth s 1*)) *else* (ϵ *x*. *x*=*x*))
by (*simp add*: *itl-def*)

lemma *finval-defs* :
$(s \models !v) =$
$(\ (if\ nfinite\ s\ then\ (v\ (nlast\ s))\ else\ (\epsilon\ \ x.\ x=x))$
)
by (*simp add*: *itl-def*)

lemma *penultval-defs* :
$(s \models v!) =$
(*if nfinite s then*
(*if nlength s* $\neq$ (*enat 0*) *then* (*v* (*nnth s* (*the-enat*(*epred*(*nlength s*))))) *else* (ϵ *x*. *x*=*x*))
else (ϵ *x*. *x*=*x*)

)
by (*simp add*: *itl-def*)

lemma *next-assign-defs* :
(*s* ⊨ *v* := *e*) = ((*if nlength s* ≠ (*enat 0*) *then v* (*nnth s 1*) *else* (ϵ *x*. *x*=*x*)) = *e s*)
by (*auto simp*: *itl-def*)

lemma *prev-assign-defs* :
(*s* ⊨ *v* =: *e*) =
(*if nfinite s then*
(*if nlength s* ≠ (*enat 0*) *then* (*v* (*nnth s* (*the-enat*(*epred*(*nlength s*))))) = *e s*
else ((ϵ *x*. *x*=*x*) = *e s*))
else True
)
by (*simp add*: *itl-def finite-defs*)

lemma *always-eqv-defs* :
(*s* ⊨ *v* ≈ *e*) =
((∀ *i*. *i*≤ *nlength s* ⟶ *v* (*nnth s i*) = *e* (*ndropn i s*))
)
by (*simp add*: *always-eq-d-def always-defs current-val-d-def ndropn-nfirst*)

lemma *temporal-assign-defs* :
(*s* ⊨ *v* ← *e*) =
(*if nfinite s then* (*v* (*nlast s*)) = *e s*
else True
)
by (*simp add*: *itl-def finite-defs*)

lemma *gets-defs* :
(*s* ⊨ *v gets e*) =
((∀ *i*. *i* < *nlength s* ⟶ *v* (*nnth s* (*Suc i*)) = *e* ((*nsubn s i* (*Suc i*))))

)
proof (*auto simp add*: *min-def finite-defs gets-d-def keep-d-def ba-defs skip-defs*
temporal-assign-d-def fin-val-d-def nsubn-nlength Suc-ile-eq nsubn-def1 ntaken-ndropn-nlast)
show ⋀*i*. ∀ *n na*. (*enat na* ≤ *nlength s* − *enat n* ⟶
enat (*n* + *na*) ≤ *nlength s* ⟶
na = *Suc 0* ⟶ *v* (*nnth s* (*Suc n*)) = *e* (*ntaken* (*Suc 0*) (*ndropn n s*))) ∧
(¬ *enat na* ≤ *nlength s* − *enat n* ⟶
enat (*n* + *na*) ≤ *nlength s* ⟶
nlength s − *enat n* = *enat* (*Suc 0*) ⟶
v (*nnth s* (*na* + *n*)) = *e* (*ntaken na* (*ndropn n s*))) ⟹
enat i < *nlength s* ⟹
v (*nnth s* (*Suc i*)) = *e* (*ntaken* (*Suc 0*) (*ndropn i s*))
by (*metis One-nat-def add.commute eSuc-enat i0-less ileI1 iless-Suc-eq ndropn-Suc-conv-ndropn*
ndropn-nlength nlength-NCons one-eSuc one-enat-def plus-1-eq-Suc zero-le)
show ⋀*n na*.
∀ *i*. *enat i* < *nlength s* ⟶ *v* (*nnth s* (*Suc i*)) = *e* (*ntaken* (*Suc 0*) (*ndropn i s*)) ⟹
¬ *na* ≤ *Suc 0* ⟹ *enat* (*n* + *na*) ≤ *nlength s* ⟹ *nlength s* − *enat n* = *enat* (*Suc 0*) ⟹

$v\ (nnth\ s\ (na + n)) = e\ (ntaken\ na\ (ndropn\ n\ s))$
by (*metis* (*no-types*) *add-diff-cancel-left′ enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat*)
qed

lemma *stable-defs-helpa*:
assumes $(\forall\, i.\ i{<}nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i))$
$i \leq nlength\ s$
shows $(v\ (nnth\ s\ i) = v\ (nfirst\ s))$
using *assms*
proof (*induct i arbitrary*:*s*)
case *0*
then show *?case* **by** (*metis ndropn-0 ndropn-nfirst*)
next
case (*Suc i*)
then show *?case*
by (*simp add*: *Suc-ile-eq*)
qed

lemma *stable-defs-helpb*:
assumes $(\forall\, i.\ i{\leq}nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
$i < nlength\ s$
shows $v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)$
using *assms*
proof (*induct i arbitrary*:*s*)
case *0*
then show *?case* **using** *Suc-ile-eq* **by** *auto*
next
case (*Suc i*)
then show *?case* **by** (*metis eSuc-enat ileI1 less-imp-le*)
qed

lemma *stable-defs-help*:
$(\forall\, i.\ i{<}nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)) =$
$(\forall\, i.\ i{\leq}nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
using *stable-defs-helpa*[*of s v*] *stable-defs-helpb*[*of s v*]
by *blast*

lemma *stable-defs*:
$(s \models stable\ v) =$
$(\forall\ i.\ i{\leq}nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))$
proof (*simp add*: *stable-d-def gets-defs current-val-d-def*)
have *1*: $\bigwedge i.\ i < nlength\ s \longrightarrow v\ (nfirst\ \ (nsubn\ s\ i\ (Suc\ i))) = v\ (nnth\ s\ i)$
by (*simp add*: *nsubn-def1 ntaken-ndropn-nfirst*)
have *2*: $(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nfirst\ (nsubn\ s\ i\ (Suc\ i)))) =$
$(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i))$
by (*simp add*: *1*)
have *3*: $(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)) =$
$(\forall\ i.\ i{\leq}nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))$
using *stable-defs-help*[*of s v*] **by** *blast*
show $(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nfirst\ (nsubn\ s\ i\ (Suc\ i)))) =$

$(\forall\, i.\ enat\ i \leq nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
by (*simp add*: *2 3*)
qed

lemma *padded-defs* :
$(s \models padded\ v) =$
$(\ ((\forall\, i.\ \ i< nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))) \vee nlength\ s = (enat\ 0))$
proof (*cases s*)
case (*NNil x1*)
then show *?thesis*
by (*auto simp add*: *padded-d-def stable-defs chop-d-def skip-defs empty-defs ntaken-nfirst ntaken-nnth zero-enat-def*)
next
case (*NCons x21 x22*)
then show *?thesis*
by (*auto simp add*: *padded-d-def stable-defs chop-d-def skip-defs empty-defs ntaken-nfirst ntaken-nnth zero-enat-def*)
(*metis One-nat-def enat.simps*(*3*) *enat-add-sub-same enat-ord-simps*(*1*) *iless-Suc-eq le-iff-add less-imp-le one-enat-def plus-1-eSuc*(*2*),
meson iless-Suc-eq less-imp-le,
metis One-nat-def enat.simps(*3*) *enat-add-sub-same enat-ord-simps*(*1*) *ile-eSuc nfinite-nlength-enat one-enat-def plus-1-eSuc*(*2*),
metis One-nat-def antisym-conv2 co.enat.sel(*2*) *diff-le-self enat-add-sub-same enat-ord-code*(*4*) *enat-ord-simps*(*1*) *epred-enat iless-Suc-eq one-enat-def plus-1-eSuc*(*2*))
qed

lemma *padded-temporal-assign-defs* :
$(s \models v <\sim e) =$
$((s \models padded\ v) \wedge$
$(if\ nfinite\ s\ then\ (v\ (nlast\ s\ )) = e\ s\ else\ True)$
$)$
by (*auto simp add*: *padded-temp-assign-d-def padded-defs temporal-assign-defs*)

lemma *chop-nfuse-1* :
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$(\sigma 1\ \models f) \wedge (\sigma 2\ \models g) \wedge$
$(nlast\ \sigma 1 = nfirst\ \sigma 2)) =$
$((\exists\ i.\ 0 \leq i \wedge i\leq nlength\ \sigma \wedge (\ ntaken\ i\ \sigma\ \models f) \wedge (ndropn\ i\ \sigma\ \models g)))$
by *auto*
(*metis enat-le-plus-same*(*1*) *nfuse-nlength ndropn-nfuse nfinite-conv-nlength-enat ntaken-nfuse the-enat.simps*,
metis nfuse-ntaken-ndropn ndropn-nfirst nfinite-ntaken ntaken-nlast)

lemma *chop-nfuse-2* :
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge$
$(nlast\ \sigma 1 = nfirst\ \sigma 2)) =$
$(\exists\ i.\ i\leq nlength\ \sigma \wedge (ntaken\ i\ \sigma)\in X \wedge (ndropn\ i\ \sigma)\in Y)$
by *auto*
(*metis enat-le-plus-same*(*1*) *ndropn-nfuse nfinite-nlength-enat nfuse-nlength ntaken-nfuse*

the-enat.simps,
metis ndropn-nfirst nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast)

lemma *chop-nfuse*:
$(\sigma \models f;g) = ($
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge (nlast\ \sigma 1 = nfirst\ \sigma 2))$
$\vee (\neg\ nfinite\ \sigma \wedge (\sigma \models f))$
$)$

by (*simp add*: *chop-defs chop-nfuse-1*)

lemmas *itl-defs = skip-defs chop-defs yields-defs infinite-defs finite-defs schop-defs syields-defs*
sometimes-defs always-defs di-defs df-defs bi-defs bf-defs da-defs ba-defs sda-defs sba-defs
next-defs wnext-defs prev-defs wprev-defs more-defs fmore-defs empty-defs init-defs
initalt-defs fin-defs finalt-defs sfin-defs halt-defs initonly-defs ifthenelse-defs
currentval-defs nextval-defs finval-defs penultval-defs next-assign-defs prev-assign-defs
always-eqv-defs temporal-assign-defs gets-defs stable-defs padded-defs
padded-temporal-assign-defs revyields-defs revsyields-defs afb-defs safb-defs

3.1.5 Soundness Axioms

ChopAssoc

lemma *ChopAssocSemHelpa*:
assumes $((\exists n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \leq nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
shows $((\exists n.\ enat\ n \leq nlength\ \sigma \wedge (\exists na{\leq}n.\ enat\ na \leq nlength\ \sigma \wedge$
$f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma))) \wedge h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists n.\ enat\ n \leq nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
proof −
have *1*: $((\exists n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \leq nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
using *assms* **by** *auto*
have *2*: $\neg\ nfinite\ \sigma \wedge f\ \sigma \Longrightarrow$
$((\exists n.\ enat\ n \leq nlength\ \sigma \wedge$
$(\exists na{\leq}n.\ enat\ na \leq nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists n.\ enat\ n \leq nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
by *simp*
have *3*: $(\exists n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \leq nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$

```
∨
        ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))) ⟹
        ((∃ n. enat n ≤ nlength σ ∧
           (∃ na≤n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
            ∧ h (ndropn n σ)) ∨
   ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
 proof −
   assume 4: (∃ n. enat n ≤ nlength σ ∧
       f (ntaken n σ) ∧
       ((∃ na. enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
∨
        ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ)))
   obtain n where 5: enat n ≤ nlength σ ∧
       f (ntaken n σ) ∧
       ((∃ na. enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
∨
        ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))
     using 4 by auto
   have 6: enat n ≤ nlength σ ∧ f (ntaken n σ)
     using 5 by auto
   have 7: ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ) ⟹
         ((∃ n. enat n ≤ nlength σ ∧
            (∃ na≤n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
             ∧ h (ndropn n σ)) ∨
        ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
     using 6 nfinite-ndropn-a by blast
   have 8: (∃ na. enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n
σ))) ⟹
           ((∃ n. enat n ≤ nlength σ ∧
              (∃ na≤n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
             ∧ h (ndropn n σ)) ∨
   ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
     proof −
       assume 9: (∃ na. enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na
(ndropn n σ)))
      show   ((∃ n. enat n ≤ nlength σ ∧
               (∃ na≤n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
                 ∧ h (ndropn n σ)) ∨
   ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
       proof −
        obtain na where 10: enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧
            h (ndropn na (ndropn n σ))
          using 9 by auto
        have 11: h (ndropn (na+n) σ)
          by (metis 10 add.commute ndropn-ndropn)
        have 12: na+n ≤ nlength σ
          by (metis 10 6 Groups.add-ac(2) dual-order.strict-implies-order enat.simps(3)
              enat-add-sub-same enat-less-enat-plusI2 le-iff-add not-le-imp-less
              order.not-eq-order-implies-strict plus-enat-simps(1))
        have 13: g (ndropn n (ntaken (n+na) σ))
```

```
      by (metis 10 12 add.commute ntaken-ndropn-swap plus-enat-simps(1))
    have 14: f (ntaken (min n (n+na)) σ)
      using 6 by linarith
    show ?thesis
    by (metis 10 12 13 14 6 add.commute le-add1 ndropn-ndropn)
   qed
  qed
 show ?thesis
 using 5 7 8 by blast
qed
show ?thesis
using 3 assms by blast
qed

lemma ChopAssocSemHelpb:
assumes ((∃ n. enat n ≤ nlength σ ∧
          (∃ na≤n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
           ∧ h (ndropn n σ)) ∨
    ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
shows ((∃ n. enat n ≤ nlength σ ∧
        f (ntaken n σ) ∧
        ((∃ na. enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
∨
         ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))) ∨
    ¬ nfinite σ ∧ f σ)
proof −
 have 1: ((∃ n. enat n ≤ nlength σ ∧
          (∃ na≤n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
           ∧ h (ndropn n σ)) ∨
    ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
  using assms by auto
 have 2: ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ)
⟹
        ((∃ n. enat n ≤ nlength σ ∧
        f (ntaken n σ) ∧
        ((∃ na. enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
∨
         ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))) ∨
    ¬ nfinite σ ∧ f σ)
  using nfinite-ndropn by blast
 have 3: (∃ n. enat n ≤ nlength σ ∧
          (∃ na≤n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
           ∧ h (ndropn n σ)) ⟹
        ((∃ n. enat n ≤ nlength σ ∧
        f (ntaken n σ) ∧
        ((∃ na. enat na ≤ nlength σ − enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
∨
         ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))) ∨
    ¬ nfinite σ ∧ f σ)
 proof −
```

assume *4*: $(\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$(\exists\, na{\leq}n.\ enat\ na \leq nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma))$
show $((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists\, na.\ enat\ na \leq nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
proof −
obtain *n* **where** *5*: $enat\ n \leq nlength\ \sigma \wedge$
$(\exists\, na{\leq}n.\ enat\ na \leq nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)$
using *4* **by** *auto*
have *6*: $(\exists\, na{\leq}n.\ enat\ na \leq nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
using *5* **by** *auto*
obtain *na* **where** *7*: $na{\leq}n \wedge enat\ na \leq nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma))$
using *6* **by** *auto*
have *8*: $na \leq nlength\ \sigma$
by (*simp add*: *7*)
have *9*: $n{-}na \leq nlength\ \sigma - na$
by (*metis 5 enat-minus-mono1 idiff-enat-enat*)
have *10*: $f\ (ntaken\ na\ \sigma)$
using *7* **by** *linarith*
have *11*: $g\ (ntaken\ (n{-}na)\ (ndropn\ na\ \sigma))$
by (*simp add*: *5 7 ntaken-ndropn-swap*)
have *12*: $h\ (ndropn\ ((n{-}na){+}na)\ \sigma)$
by (*simp add*: *5 7*)
have *13*: $h\ (ndropn\ (n{-}na)\ (ndropn\ na\ \sigma))$
by (*metis 12 add.commute ndropn-ndropn*)
show *?thesis*
using *10 11 13 7 9* **by** *auto*
qed
qed
show *?thesis*
using *2 3 assms* **by** *blast*
qed

lemma *ChopAssocSemHelp*:
$((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists\, na.\ enat\ na \leq nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma) =$
$((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$(\exists\, na{\leq}n.\ enat\ na \leq nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$

using *ChopAssocSemHelpa*[*of* σ *f g h*] *ChopAssocSemHelpb*[*of* σ *f g h*] **by** *blast*

lemma *ChopAssocSemHelp1*:
$((\ \sigma) \models\ f\ ;\ (g\ ;\ h)) = ((\ \sigma) \models (f;g);h)$
proof −
have $(\sigma \models\ f\ ;\ (g\ ;\ h)) = ((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists\, na.\ enat\ na \leq nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
by (*simp add*: *chop-defs*)
also have ... =
$((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$(\exists\, na{\leq}n.\ enat\ na \leq nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
using *ChopAssocSemHelp*[*of* σ *f g h*] **by** *blast*
also have ... =
$(\ \sigma \models (f;g);h)$ **by** (*simp add*: *chop-defs*)
finally show $(\ \sigma \models\ f\ ;\ (g\ ;\ h)) = (\ \sigma \models (f;g);h)$ **.**
qed

lemma *ChopAssocSem*:
$(\sigma \models\ f\ ;\ (g\ ;\ h) =\ (f;g);h)$
using *ChopAssocSemHelp1*[*of f g h* σ] **by** *auto*

OrChopImp

lemma *OrChopImpSem*:
$(\sigma \models\ (\ f \vee g);h\ \longrightarrow\ f;h \vee g;h\ )$
by (*auto simp add*: *chop-defs*)

ChopOrImp

lemma *ChopOrImpSem*:
$(\sigma \models f;(g \vee h) \longrightarrow\ \ f;g \vee\ f;h\ )$
by (*auto simp add*: *chop-defs*)

EmptyChop

lemma *EmptyChopSem*:
$(\sigma \models empty\ ;\ f = f\ )$
by (*simp add*: *chop-defs empty-defs min-def*)
(*metis enat-0-iff*(*1*) *ndropn-0 nlength-eq-enat-nfiniteD zero-le*)

ChopEmpty

lemma *ChopEmptySem*:
$(\sigma \models f;empty = f\ )$
by (*simp add*: *chop-defs empty-defs min-def*)

(*metis cancel-comm-monoid-add-class.diff-cancel enat-diff-cancel-left idiff-enat-enat nfinite-nlength-enat ntaken-all order-refl zero-enat-def*)

StateImpBi

lemma *StateImpBiSem*:
$(\sigma \models init\ f \longrightarrow \quad bi\ (init\ f)\ )$
by (*simp add*: *init-defs bi-defs*)

NextImpNotNextNot

lemma *NextImpNotNextNotSem*:
$(\sigma \models \bigcirc\ f \longrightarrow\ \neg\ (\bigcirc\ (\neg\ f))\ )$
by (*simp add*: *next-defs*)

BiBoxChopImpChop

lemma *BiBoxChopImpChopSem*:
$(\sigma \models bi\ (\ f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1\ )$
by (*simp add*: *bi-defs always-defs chop-defs*)
fastforce

BoxInduct

lemma *box-induct-help-1* :
$\bigwedge j.\ \forall\, n.\ enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$
$nlength\ \sigma - enat\ n = (enat\ 0) \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \Longrightarrow$
$f\ \sigma \Longrightarrow$
$enat\ j \leq nlength\ \sigma \Longrightarrow$
$f\ (ndropn\ j\ \sigma)$
proof −
fix *j*
show $\forall\, n.\ enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$
$nlength\ \sigma - enat\ n = (enat\ 0) \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \Longrightarrow$
$f\ \sigma \Longrightarrow$
$enat\ j \leq nlength\ \sigma \Longrightarrow$
$f\ (ndropn\ j\ \sigma)$
proof (*induct j arbitrary*: σ)
case *0*
then show *?case* **by** *simp*
next
case (*Suc j*)
then show *?case* **by** (*simp add*: *ndropn-ndropn*)
(*metis Suc-ile-eq add.right-neutral enat.simps*(*3*) *enat-add-sub-same le-cases le-iff-add not-less zero-enat-def*)
qed
qed

lemma *BoxInductSem*:
$(\sigma \models \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f)$
proof

(*auto simp add*: *always-defs wnext-defs*)
show $\bigwedge n.\ \forall\, n.\ enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$
$nlength\ \sigma - enat\ n = enat\ 0 \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \Longrightarrow$
$f\ \sigma \Longrightarrow$
$enat\ n \leq nlength\ \sigma \Longrightarrow$
$f\ (ndropn\ n\ \sigma)$
using *box-induct-help-1* **by** *blast*
qed

3.1.6 Quantification over State (Flexible) Variables

Quantification in Infinite ITL is done similarly as in Finite ITL.

typedecl *state*

instance *state* :: *world* **..**

type-synonym $'a$ *statefun* = (*state*,$'a$) *stfun*
type-synonym *statepred* = *bool statefun*
type-synonym $'a$ *tempfun* = (*state*,$'a$) *formfun*
type-synonym *temporal* = *state formula*

3.1.7 Temporal Quantifiers

definition *exist-state-d* :: ($'a$ *statefun* ⇒ *temporal*)⇒ *temporal* (**binder** *Eex* *10*)
where *exist-state-d* *F* ≡ (λ*s*. (∃ *x*. *s* ⊨ *F x*))

syntax
-Eex :: [*idts*, *lift*] ⇒ *lift* ((*3*∃∃ *-./ -*) [*0*,*10*] *10*)

translations
-Eex v A == *Eex v*. *A*

definition *forall-state-d* :: ($'a$ *statefun* ⇒ *temporal*)⇒ *temporal* (**binder** *Aall* *10*)
where *forall-state-d F* ≡ *LIFT*(¬(∃∃ *x*. ¬(*F x*)))

syntax
-Aall :: [*idts*, *lift*] ⇒ *lift* ((*3*∀∀ *-./ -*) [*0*,*10*] *10*)

translations
-Aall v A == *Aall v*. *A*

3.1.8 bind

definition *sbind* :: *temporal* ⇒ (*bool statefun* ⇒ *temporal*) ⇒ *temporal*
where *sbind F1 F2* ≡ (λ*s*. (∃ *v*. *s* ⊨ □($*v* = *F1*) ∧ *F2 v*))

definition *pbind* :: *temporal* ⇒ (*bool statefun* ⇒ *temporal*) ⇒ *temporal*
where *pbind F1 F2* ≡ (λ*s*. (∃ *v*. *s* ⊨ *bf*(!*v* = *F1*) ∧ *F2 v*))

syntax
-sbind :: [*idt*, *lift*, *lift*] ⇒ *lift* ((*3sfxbnd -/ ≃ -/ : -*) [*0*,*10*,*10*] *10*)
-pbind :: [*idt*, *lift*, *lift*] ⇒ *lift* ((*3pfxbnd -/ ≃ -/ : -*) [*0*,*10*,*10*] *10*)

syntax (*ASCII*)
-sbind :: [*idt*, *lift*, *lift*] ⇒ *lift* ((*3sfxbnd -/ as -/ : -*) [*0*,*10*,*10*] *10*)
-pbind :: [*idt*, *lift*, *lift*] ⇒ *lift* ((*3pfxbnd -/ as -/ : -*) [*0*,*10*,*10*] *10*)

translations
-sbind v F1 F2 == *CONST sbind F1* (λ *v*. *F2*)
-pbind v F1 F2 == *CONST pbind F1* (λ *v*. *F2*)

lemma *sfxbnd-defs*:
(σ ⊨ (*sfxbnd v ≃ f1* : (*f2 v*))) =
(∃ *v*. (∀ *n*. *enat n* ≤ *nlength* σ ⟶ *v* (*nnth* σ *n*) = *f1* (*ndropn n* σ)) ∧ *f2 v* σ)
by (*simp add*: *sbind-def itl-defs ndropn-nfirst*)

lemma *pfxbnd-defs*:
(σ ⊨ (*pfxbnd v ≃ f1* : (*f2 v*))) =
(∃ *v*. (∀ *n*. *enat n* ≤ *nlength* σ ⟶ *v* (*nnth* σ *n*) = *f1* (*ntaken n* σ)) ∧ *f2 v* σ)
by (*simp add*: *pbind-def itl-defs ntaken-nlast*)

end

3.2 Axioms and Rules

theory *ITL*
imports
Semantics
begin

The Finite and Infinite ITL axiom and proof rules are introduced (taken from [15]). The soundness of the rules and axioms are checked using the lemmas of Semantics.thy.

3.2.1 Rules

lemma *MP* :
assumes ⊢ *f* ⟶ *g*
⊢ *f*
shows ⊢ *g*
using *assms* **by** *fastforce*

lemma *BoxGen* :
assumes ⊢ *f*
shows ⊢ □ *f*
using *assms*
by (*auto simp add*: *itl-defs Valid-def*)

lemma *BiGen*:
assumes $\vdash f$
shows $\vdash bi\ f$
using *assms*
by (*auto simp add*: *itl-defs Valid-def*)

3.2.2 Axioms

lemma *ChopAssoc* :
$\vdash\ f\ ;\ (g\ ;\ h) =\ (f;g);h$
using *ChopAssocSem Valid-def* **by** *blast*

lemma *OrChopImp* :
$\vdash\ (\ f \vee g);h \longrightarrow\ f;h \vee g;h$
using *OrChopImpSem Valid-def* **by** *blast*

lemma *ChopOrImp* :
$\vdash\ f;(g \vee h) \longrightarrow\ f;g \vee\ f;h$
using *ChopOrImpSem Valid-def* **by** *blast*

lemma *EmptyChop* :
$\vdash\ empty\ ;\ f = f$
using *EmptyChopSem Valid-def* **by** *blast*

lemma *ChopEmpty* :
$\vdash f;empty = f$
using *ChopEmptySem Valid-def* **by** *blast*

lemma *StateImpBi* :
$\vdash init\ f \longrightarrow\ bi\ (init\ f)$
using *StateImpBiSem Valid-def* **by** *blast*

lemma *NextImpNotNextNot* :
$\vdash\ \bigcirc f \longrightarrow\ \neg\ (\bigcirc\ (\neg\ f))$
using *NextImpNotNextNotSem Valid-def* **by** *blast*

lemma *BiBoxChopImpChop* :
$\vdash\ bi\ (\ f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1$
using *BiBoxChopImpChopSem Valid-def* **by** *blast*

lemma *BoxInduct* :
$\vdash \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f$
using *BoxInductSem Valid-def* **by** *blast*

3.2.3 Additional Lemmas

The following is again from [9, 4] but adapted for our need.

lemma *int-eq-true*:
assumes $\vdash P$
shows $\vdash P = \#True$

using *assms* **by** *auto*

lemma *int-eq*:
assumes $\vdash X = Y$
shows $X = Y$
using *assms* **by** (*auto simp*: *inteq-reflection*)

lemma *int-iffI*:
assumes $\vdash F \longrightarrow G$ **and** $\vdash G \longrightarrow F$
shows $\vdash F = G$
using *assms* **by** *force*

lemma *int-iffD1*: **assumes** *h*: $\vdash F = G$ **shows** $\vdash F \longrightarrow G$
using *h* **by** *auto*

lemma *int-iffD2*: **assumes** *h*: $\vdash F = G$ **shows** $\vdash G \longrightarrow F$
using *h* **by** *auto*

lemma *lift-imp-trans*:
assumes $\vdash A \longrightarrow B$ **and** $\vdash B \longrightarrow C$
shows $\vdash A \longrightarrow C$
using *assms* **by** *force*

lemma *lift-imp-neg*: **assumes** $\vdash A \longrightarrow B$ **shows** $\vdash \neg B \longrightarrow \neg A$
using *assms* **by** *auto*

lemma *lift-and-com*: $\vdash (A \wedge B) = (B \wedge A)$
by *auto*

3.2.4 Quantification

lemma *EExI* :
$\vdash F\ y \longrightarrow (\exists\exists\ x\ .\ F\ x)$
by (*auto simp add*: *exist-state-d-def Valid-def*)

lemma *EExE*:
assumes $\bigwedge x. \vdash F\ x \longrightarrow G$
shows $\vdash (\exists\exists\ x.\ F\ x) \longrightarrow G$
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *Valid-def exist-state-d-def unl-lift2*)

lemma *EExVal*:
$((\ w) \models (\exists\exists\ x.\ F\ x)) =$
$(\exists\ x\ (val :: 'a\ nellist).\ (\ (\ val = (nmap\ x\ w) \wedge\ ((\ w) \models F\ x))))$
by (*simp add*: *exist-state-d-def*)

lemma *AAxDef*:
$\vdash (\forall\forall\ x.\ F\ x) = (\neg(\exists\exists\ x.\ \neg(F\ x)))$
by (*simp add*: *Valid-def forall-state-d-def exist-state-d-def*)

lemma *ExEqvRule*:
assumes $\bigwedge x. \vdash (f\ x) = (g\ x)$
shows $\vdash (\exists\ x.\ f\ x) = (\exists\ x.\ g\ x)$
using *assms* **by** *fastforce*

3.2.5 Lemmas about *current-val*

lemma *current-const*: $\vdash \$(\#c) = \#c$
by (*simp add*: *current-val-d-def intI*)

lemma *current-fun1*: $\vdash \$(f{<}x{>}) = f\ {<}\$x{>}$
by (*simp add*: *current-val-d-def intI*)

lemma *current-fun2*: $\vdash \$(f{<}x,y{>}) = f\ {<}\$x,\$y{>}$
by (*auto simp*: *current-val-d-def intI*)

lemma *current-fun3*: $\vdash \$(f{<}x,y,z{>}) = f\ {<}\$x,\$y,\$z{>}$
by (*auto simp*: *current-val-d-def intI*)

lemma *current-forall*: $\vdash \$(\forall\ x.\ P\ x) = (\forall\ x.\ \$(P\ x))$
by (*auto simp*: *current-val-d-def intI*)

lemma *current-exists*: $\vdash \$(\exists\ x.\ P\ x) = (\exists\ x.\ \$(P\ x))$
by (*auto simp*: *current-val-d-def intI*)

lemma *current-exists1*: $\vdash \$(\exists!\ x.\ P\ x) = (\exists!\ x.\ \$(P\ x))$
by (*auto simp*: *current-val-d-def intI*)

lemmas *all-current* = *current-const current-fun1 current-fun2 current-fun3*
current-forall current-exists current-exists1

lemmas *all-current-unl* = *all-current*[*THEN intD*]
lemmas *all-current-eq* = *all-current*[*THEN inteq-reflection*]

3.2.6 Lemmas about *next-val*

lemma *next-const*: $\vdash more \longrightarrow (\#c)\$ = \#c$
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun1*: $\vdash more \longrightarrow f{<}x{>}\$ = f{<}x\${>}$
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun2*: $\vdash more \longrightarrow f{<}x,y{>}\$ = f\ {<}x\$,y\${>}$
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun3*: $\vdash more \longrightarrow f{<}x,y,z{>}\$ = f\ {<}x\$,y\$,z\${>}$
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-forall*: $\vdash more \longrightarrow (\forall\ x.\ P\ x)\$ = (\forall\ x.\ (P\ x)\$)$
by (*auto simp*: *next-val-d-def intI*)

lemma *next-exists*: $\vdash$ *more* $\longrightarrow$ $(\exists\ x.\ P\ x)\$ = (\exists\ x.\ (P\ x)\$)$
by (*auto simp*: *next-val-d-def intI*)

lemma *next-exists1*: $\vdash$ *more* $\longrightarrow$ $(\exists!\ x.\ P\ x)\$ = (\exists!\ x.\ (P\ x)\$)$
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemmas *all-next* = *next-const next-fun1 next-fun2 next-fun3*
next-forall next-exists next-exists1

lemmas *all-next-unl* = *all-next*[*THEN intD*]

3.2.7 Lemmas about *fin-val*

lemma *fin-const*: $\vdash$ *finite* $\longrightarrow$ $!(\#c) = \#c$
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-fun1*: $\vdash$ *finite* $\longrightarrow$ $!(f<x>) = f\ <!x>$
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-fun2*: $\vdash$ *finite* $\longrightarrow$ $!(f<x,y>) = f\ <!x,\ !y>$
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-fun3*: $\vdash$ *finite* $\longrightarrow$ $!(f<x,y,z>) = f\ <!x,!y,!z>$
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-forall*: $\vdash$ *finite* $\longrightarrow$ $!(\forall\ x.\ P\ x) = (\forall\ x.\ !(P\ x))$
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-exists*: $\vdash$ *finite* $\longrightarrow$ $!(\exists\ x.\ P\ x) = (\exists\ x.\ !(P\ x))$
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-exists1*: $\vdash$ *finite* $\longrightarrow$ $!(\exists!\ x.\ P\ x) = (\exists!\ x.\ !(P\ x))$
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemmas *all-fin* = *fin-const fin-fun1 fin-fun2 fin-fun3*
fin-forall fin-exists fin-exists1

lemmas *all-fin-unl* = *all-fin*[*THEN intD*]

3.2.8 Lemmas about *penult-val*

lemma *penult-const*: $\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ $(\#c)! = \#c$
by (*auto simp*: *penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-fun1*: $\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ $f<x>! = f<x!>$
by (*auto simp*: *penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-fun2*: $\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ $f<x,y>! = f <x!,y!>$
by (*auto simp*: *penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-fun3*: $\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ $f<x,y,z>! = f <x!,y!,z!>$
by (*auto simp*: *penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-forall*: $\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ $(\forall\ x.\ P\ x)! = (\forall\ x.\ (P\ x)!)$
by (*auto simp*: *penult-val-d-def finite-defs intI*)

lemma *penult-exists*: $\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ $(\exists\ x.\ P\ x)! = (\exists\ x.\ (P\ x)!)$
by (*auto simp*: *penult-val-d-def finite-defs intI*)

lemma *penult-exists1*: $\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ $(\exists!\ x.\ P\ x)! = (\exists!\ x.\ (P\ x)!)$
by (*auto simp*: *penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemmas *all-penult* = *penult-const penult-fun1 penult-fun2 penult-fun3*
penult-forall penult-exists penult-exists1

lemmas *all-penult-unl* = *all-penult*[*THEN intD*]

3.2.9 Basic temporal variables properties

lemma *empty-imp-fin-eqv-curr*:
$\vdash$ *empty* $\longrightarrow$ $!v = \$v$
by (*simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD*)
(*metis nlast-NNil nlength-eq-enat-nfiniteD nnth-nlast ntaken-0 ntaken-nlast the-enat-0 zero-enat-def*)

lemma *skip-imp-fin-eqv-next*:
$\vdash$ *skip* $\longrightarrow$ $!v = v\$$
by (*simp add*: *Valid-def itl-defs*)
(*metis One-nat-def le-numeral-extra*(*4*) *ndropn-0 nlength-eq-enat-nfiniteD*
ntaken-all ntaken-ndropn-nlast one-enat-def plus-1-eq-Suc)

lemma *skip-imp-penult-eqv-curr*:
$\vdash$ *skip* $\longrightarrow$ $v! = \$v$
by (*simp add*: *Valid-def itl-defs current-val-d-def nlength-eq-enat-nfiniteD*)
(*metis ndropn-0 ndropn-nfirst*)

end

3.3 Weak chop operator

theory *Theorems*
imports
ITL
begin

We give the proofs of a list of Finite and Infinite ITL theorems. These proofs and theorems are from

[14] but adapted for infinite and finite intervals.

3.3.1 Propositional reasoning

This is a list of propositional logic theorems used in the proofs of the ITL theorems.

lemma *IfThenElseImp*:
$\vdash (if_i \;\; g \;\; then \;\; f \;\; else \; f1) \longrightarrow ((\; g \longrightarrow f) \wedge (\neg g \longrightarrow f1))$
by (*simp add*: *itl-def Valid-def*)

lemma *Prop01*:
assumes $\vdash f \longrightarrow \neg\; g \vee h$
shows $\vdash g \wedge f \longrightarrow h$
using *assms* **by** *auto*

lemma *Prop02*:
assumes $\vdash f \longrightarrow g$
$\vdash f1 \longrightarrow g$
shows $\vdash f \vee f1 \longrightarrow g$
using *assms* **by** *fastforce*

lemma *Prop03*:
assumes $\vdash f = (g \vee h)$
shows $\vdash h \longrightarrow f$
using *assms* **by** *auto*

lemma *Prop04*:
assumes $\vdash f = h$
$\vdash f = h1$
shows $\vdash h1 = h$
using *assms* **using** *int-eq* **by** *auto*

lemma *Prop05*:
assumes $\vdash f \longrightarrow g$
shows $\vdash f \longrightarrow h \vee g$
using *assms* **by** *auto*

lemma *Prop06*:
assumes $\vdash f = (g \vee h)$
$\vdash h = h1$
shows $\vdash f = (g \vee h1)$
using *assms* **by** *fastforce*

lemma *Prop07*:
assumes $\vdash f \longrightarrow g \vee h$
shows $\vdash f \wedge \neg\; g \longrightarrow h$
using *assms* **by** *auto*

lemma *Prop08*:
assumes $\vdash f \longrightarrow g \vee h$

$\vdash h \longrightarrow h1$
shows $\vdash f \longrightarrow g \vee h1$
using *assms* **by** *fastforce*

lemma *Prop09*:
assumes $\vdash f \wedge g \longrightarrow h$
shows $\vdash f \longrightarrow (g \longrightarrow h)$
using *assms* **by** *auto*

lemma *Prop10*:
assumes $\vdash f \longrightarrow g$
shows $\vdash f = (f \wedge g)$
using *assms* **by** *auto*

lemma *Prop11*:
$(\vdash f = f1) = (\ (\vdash f \longrightarrow f1) \wedge\ (\vdash f1 \longrightarrow f)\ )$
by (*auto simp*: *Valid-def*)

lemma *Prop12*:
$(\vdash f\ \longrightarrow (\ f1 \wedge f2)) = (\ (\vdash f \longrightarrow f1) \wedge (\vdash f \longrightarrow f2))$
by (*auto simp*: *Valid-def*)

lemma *Prop13*:
assumes $\vdash f \longrightarrow g \vee h$
shows $\vdash f \wedge \neg h \longrightarrow g$
using *assms* **by** (*auto simp*: *Valid-def*)

3.3.2 State formulas

The *init* operator denotes state formulas, i.e., ITL formula that only constrain the first state of an interval.

lemma *Initprop* :
$\vdash ((init\ f) \wedge (init\ g)) = init(f \wedge g)$
$\vdash (\neg\ (init\ f)) = init\ (\ \neg\ f\ )$
$\vdash ((init\ f) \vee (init\ g)) = init\ (f \vee g)$
$\vdash init\ \#True$
by (*auto simp*: *itl-defs*)

lemma *Finprop* :
$\vdash ((\#True;(f \wedge empty)) \wedge (\#True;(g \wedge empty))) = (\#True;((f \wedge g) \wedge empty))$
$\vdash ((\#True;(f \wedge empty)) \vee (\#True;(g \wedge empty))) = (\#True;((f \vee g) \wedge empty))$
$\vdash (\#True;((\#True) \wedge empty))$
$\vdash finite \longrightarrow (\neg\ (\#True;(f \wedge empty))) = (\#True;(\neg f \wedge empty))$
$\vdash (\neg\ (\#True;(f \wedge empty))) = (\ (\#True;(\neg f \wedge empty)) \wedge finite)$
using *nfinite-nlength-enat*
by (*auto simp*: *finalt-defs finite-defs zero-enat-def*,
auto simp add: *itl-defs nfinite-nlength-enat zero-enat-def*, *force*)

3.3.3 finite and inf properties

lemma *EmptyImpFinite*:
$\vdash$ *empty* $\longrightarrow$ *finite*
using *nlength-eq-enat-nfiniteD* **by** (*auto simp add*: *itl-defs zero-enat-def*)

lemma *SkipChopFiniteImpFinite*:
$\vdash$ *skip*;*finite* $\longrightarrow$ *finite*
using *nfinite-ndropn nlength-eq-enat-nfiniteD*
by (*simp add*: *Valid-def itl-defs*,*force*)

lemma *FiniteChopSkipImpFinite*:
$\vdash$ *finite*;*skip* $\longrightarrow$ *finite*
using *nlength-eq-enat-nfiniteD*
by (*simp add*: *Valid-def itl-defs*, *force*)

lemma *FiniteChopSkipImpMore*:
$\vdash$ *finite*;*skip* $\longrightarrow$ *more*
using *nlength-eq-enat-nfiniteD one-enat-def*
by (*simp add*: *Valid-def itl-defs*, *force*)

lemma *FiniteAndMoreImpFiniteChopSkip*:
$\vdash$ *finite* $\wedge$ *more* $\longrightarrow$ *finite*;*skip*
by (*auto simp add*: *Valid-def itl-defs zero-enat-def*)
(*metis Suc-ile-eq diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *idiff-enat-enat nfinite-nlength-enat*)

lemma *FiniteChopSkipEqvFiniteAndMore*:
$\vdash$ *finite*;*skip* = (*finite* $\wedge$ *more*)
by (*simp add*: *FiniteAndMoreImpFiniteChopSkip FiniteChopSkipImpFinite FiniteChopSkipImpMore*
Prop12 int-iffI)

lemma *FiniteChopSkipEqvSkipChopFinite*:
$\vdash$ *finite*;*skip* = *skip*;*finite*
by (*auto simp add*: *Valid-def itl-defs*)
(*metis enat.distinct*(*1*) *enat-add-sub-same enat-le-plus-same*(*2*) *le-iff-add* ,
metis eSuc-enat enat.simps(*3*) *enat-add-sub-same idiff-enat-0-right iless-Suc-eq le-zero-eq*
less-eqE min.orderE nlength-eq-enat-nfiniteD not-le one-eSuc plus-1-eSuc(*2*),
metis add.commute enat.simps(*3*) *enat-add-sub-same idiff-enat-enat le-iff-add*
nfinite-nlength-enat)

lemma *FiniteAndEmptyEqvEmpty*:
$\vdash$ (*finite* $\wedge$ *empty*) = *empty*
by (*auto simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD*
zero-enat-def)

lemma *FiniteChopFiniteEqvFinite*:
$\vdash$ *finite*;*finite* = *finite*
by (*auto simp add*: *Valid-def itl-defs*) (*metis zero-enat-def zero-le*)

lemma *InfChopInfEqvInf*:
$\vdash$ *inf*;*inf* = *inf*

by (*simp add*: *Valid-def itl-defs*)

lemma *InfChopFiniteEqvInf*:
$\vdash$ *inf*;*finite* = *inf*
by (*simp add*: *Valid-def itl-defs*)

lemma *FiniteChopInfEqvInf*:
$\vdash$ *finite*;*inf* = *inf*
by (*auto simp add*: *Valid-def itl-defs*) (*metis zero-enat-def zero-le*)

lemma *InfEqvNotFinite*:
$\vdash$ *inf* = ($\neg$ *finite*)
by (*simp add*: *Valid-def itl-defs*)

lemma *FiniteEqvNotInf*:
$\vdash$ *finite* = ($\neg$ *inf*)
by (*simp add*: *Valid-def itl-defs*)

lemma *ChopTrueAndFiniteEqvAndFiniteChopFinite*:
$\vdash$ ((*f*;#*True*) $\wedge$ *finite*) = (*f* $\wedge$ *finite*);*finite*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *TrueChopAndFiniteEqvAndFiniteChopFinite*:
$\vdash$ ((#*True*;*f*) $\wedge$ *finite*) = *finite*;(*f* $\wedge$ *finite*)
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteChopMoreEqvMore*:
$\vdash$ *finite*;*more* = *more*
by (*auto simp add*: *Valid-def itl-defs*)
(*metis idiff-0-right zero-enat-def zero-le*)

lemma *ChopAndFiniteDist*:
$\vdash$ ((*f*;*g*) $\wedge$ *finite*) = (*f* $\wedge$ *finite*);(*g* $\wedge$ *finite*)
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteOrInfinite*:
$\vdash$ *finite* $\vee$ *inf*
by (*simp add*: *Valid-def itl-defs*)

lemma *FiniteImpAnd*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* = *g*
shows $\vdash$ (*f* $\wedge$ *finite*) = (*g* $\wedge$ *finite*)
using *assms* **by** (*auto simp add*: *Valid-def itl-defs*)

lemma *FmoreEqvSkipChopFinite*:
$\vdash$ *fmore* = *skip*;*finite*
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
fmore-d-def inteq-reflection lift-and-com)

lemma *FiniteImp*:

$\vdash (f \wedge \mathit{finite} \longrightarrow g) = (f \wedge \mathit{finite} \longrightarrow g \wedge \mathit{finite})$
by (*simp add*: *itl-defs Valid-def*)

lemma *ChopAndInf*:
$\vdash ((f;g) \wedge \mathit{inf}) = (f;(g \wedge \mathit{inf}))$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *ChopAndInfB*:
$\vdash ((f;g) \wedge \mathit{inf}) = ((f \wedge \mathit{inf}) \vee (f \wedge \mathit{finite});(g \wedge \mathit{inf})\ \ )$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *MoreAndInfEqvInf*:
$\vdash (\mathit{more} \wedge \mathit{inf}) = \mathit{inf}$
by (*metis ChopAndInf EmptyImpFinite FiniteChopMoreEqvMore InfEqvNotFinite Prop11 Prop12 empty-d-def finite-d-def int-simps*(*32*) *inteq-reflection*)

lemma *AndInfChopAndInfEqvAndInf*:
$\vdash (f \wedge \mathit{inf});(f \wedge \mathit{inf}) = (f \wedge \mathit{inf})$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndInfChopEqvAndInf*:
$\vdash (f \wedge \mathit{inf});g = (f \wedge \mathit{inf})$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndMoreAndInfEqvAndInf*:
$\vdash ((f \wedge \mathit{more}) \wedge \mathit{inf}) = (f \wedge \mathit{inf})$
by (*auto simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)
(*metis gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *AndMoreAndFiniteEqvAndFmore*:
$\vdash ((f \wedge \mathit{more}) \wedge \mathit{finite}) = (f \wedge \mathit{fmore})$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *NotFmoreAndEmpty*:
$\vdash \neg\ (\mathit{empty} \wedge \mathit{fmore})$
by (*auto simp add*: *fmore-d-def empty-d-def*)

lemma *NotFmoreAndInf*:
$\vdash \neg\ ((f \wedge \mathit{inf}) \wedge \mathit{fmore})$
by (*auto simp add*: *fmore-d-def finite-d-def infinite-d-def*)

lemma *FmoreChopAnd*:
$\vdash (((f \wedge \mathit{more});g) \wedge \mathit{fmore}) = ((f \wedge \mathit{fmore});(g \wedge \mathit{finite}))$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *NotEmptyAndInf*:
$\vdash \neg(\mathit{empty} \wedge \mathit{inf})$
by (*auto simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *OrFiniteInf*:

$\vdash\ f = ((\ f \wedge \mathit{finite}) \vee (\ f \wedge \mathit{inf}))$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndInfEqvChopFalse*:
$\vdash (f \wedge \mathit{inf}) = f;\#\mathit{False}$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *EmptyOrMoreSplit*:
$\vdash f = ((f \wedge \mathit{empty}) \vee (f \wedge \mathit{more}))$
unfolding *empty-d-def* **by** *auto*

3.3.4 Basic Theorems

lemma *BiChopImpChop* :
$\vdash \mathit{bi}\ (f \longrightarrow \mathit{f1}) \longrightarrow f;g \longrightarrow \mathit{f1};g$
proof −
have *1*: $\vdash g \longrightarrow g$ **by** *auto*
hence *2*: $\vdash \Box\ (\ g \longrightarrow g)$ **by** (*rule BoxGen*)
have *3*: $\vdash \mathit{bi}\ (\ f \longrightarrow \mathit{f1}) \wedge \Box(g \longrightarrow g) \longrightarrow f;g \longrightarrow \mathit{f1};g$ **by** (*rule BiBoxChopImpChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *AndChopA*:
$\vdash (f \wedge \mathit{f1});g \longrightarrow f;g$
proof −
have *1*: $\vdash f \wedge \mathit{f1} \longrightarrow f$ **by** *auto*
hence *2*: $\vdash \mathit{bi}\ (f \wedge \mathit{f1} \longrightarrow f)$ **by** (*rule BiGen*)
have *3*: $\vdash \mathit{bi}\ (f \wedge \mathit{f1} \longrightarrow f) \longrightarrow (f\ \wedge \mathit{f1});g \longrightarrow f;g$ **by** (*rule BiChopImpChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *AndChopB*:
$\vdash (f \wedge \mathit{f1});g \longrightarrow \mathit{f1};g$
proof −
have *1*: $\vdash f \wedge \mathit{f1} \longrightarrow \mathit{f1}$ **by** *auto*
hence *2*: $\vdash\ \mathit{bi}\ (f \wedge \mathit{f1} \longrightarrow \mathit{f1})$ **by** (*rule BiGen*)
have *3*: $\vdash\ \mathit{bi}\ (f \wedge \mathit{f1} \longrightarrow \mathit{f1}) \longrightarrow (f \wedge \mathit{f1});g \longrightarrow \mathit{f1};g$ **by** (*rule BiChopImpChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *NextChop*:
$\vdash (\bigcirc f);g\ = \bigcirc(f;g)$
proof −
have *1*: $\vdash \mathit{skip};(f;g)\ = (\mathit{skip};f);g$ **by** (*rule ChopAssoc*)
show *?thesis* **by** (*metis 1 int-eq next-d-def*)
qed

lemma *BoxChopImpChop* :
$\vdash\ \Box\ (g \longrightarrow \mathit{g1}) \longrightarrow f;g \longrightarrow f;\mathit{g1}$

proof −
have *1*: ⊢ *g* ⟶ *g* **by** *auto*
hence *2*: ⊢ *bi* (*g* ⟶ *g*) **by** (*rule BiGen*)
have *3*: ⊢ *bi* (*f* ⟶ *f*) ∧ □(*g* ⟶ *g1*) ⟶ *f*;*g* ⟶ *f*;*g1* **by** (*rule BiBoxChopImpChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftChopImpChop*:
assumes ⊢ *f*⟶ *f1*
shows ⊢ *f*;*g* ⟶ *f1*;*g*
proof −
have *1*: ⊢ *f*⟶ *f1* **using** *assms* **by** *auto*
hence *2*: ⊢ *bi* (*f* ⟶ *f1*) **by** (*rule BiGen*)
have *3*: ⊢ *bi* (*f* ⟶ *f1*) ⟶ *f*;*g* ⟶ *f1*;*g* **by** (*rule BiChopImpChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightChopImpChop*:
assumes ⊢ *g* ⟶ *g1*
shows ⊢ *f*;*g* ⟶ *f*;*g1*
proof −
have *1*: ⊢ *g* ⟶ *g1* **using** *assms* **by** *auto*
hence *2*: ⊢ □ (*g* ⟶ *g1*) **by** (*rule BoxGen*)
have *3*: ⊢ □ (*g* ⟶ *g1*) ⟶ *f*;*g* ⟶ *f*;*g1* **by** (*rule BoxChopImpChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightChopEqvChop*:
assumes ⊢ *g* = *g1*
shows ⊢ (*f*;*g*) = (*f*;*g1*)
using *assms RightChopImpChop*[*of g g1 f*] *RightChopImpChop*[*of g1 g f*]
by *fastforce*

lemma *InfRightChopEqvChop*:
assumes ⊢ *inf* ⟶ *g* = *g1*
shows ⊢ *inf* ⟶ (*f*;*g*) = (*f*;*g1*)
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *ChopOrEqv*:
⊢ *f*;(*g* ∨ *g1*) = (*f*;*g* ∨ *f*;*g1*)
proof −
have *1*: ⊢ *g* ⟶ *g* ∨ *g1* **by** *auto*
hence *2*: ⊢ *f*;*g* ⟶ *f*;(*g* ∨ *g1*) **by** (*rule RightChopImpChop*)
have *3*: ⊢ *g1* ⟶ *g* ∨ *g1* **by** *auto*
hence *4*: ⊢ *f*;*g1* ⟶ *f*;(*g* ∨ *g1*) **by** (*rule RightChopImpChop*)
from *2 4* **show** *?thesis* **by** (*meson ChopOrImp Prop02 Prop11*)
qed

lemma *OrChopEqv*:

$\vdash (f \vee f1);g = (f;g \vee f1;g)$
proof −
have *1*: $\vdash f \longrightarrow f \vee f1$ **by** *auto*
hence *2*: $\vdash f;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*
hence *4*: $\vdash f1;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
from *2 4* **show** *?thesis*
by (*meson OrChopImp int-iffI Prop02*)
qed

lemma *OrChopImpRule*:
assumes $\vdash f \longrightarrow f1 \vee f2$
shows $\vdash f;g \longrightarrow (f1;g) \vee (f2;g)$
proof −
have *1*: $\vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;g \longrightarrow (f1 \vee f2);g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash (f1 \vee f2);\ g = (f1;g \vee f2;g)$ **by** (*rule OrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftChopEqvChop*:
assumes $\vdash f = f1$
shows $\vdash f;g = (f1;g)$
proof −
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow f1$ **by** *auto*
hence *3*: $\vdash f;g \longrightarrow f1;g$ **by** (*rule LeftChopImpChop*)
have $\vdash f1 \longrightarrow f$ **using** *1* **by** *auto*
hence *4*: $\vdash f1;g \longrightarrow f;g$ **by** (*rule LeftChopImpChop*)
from *3 4* **show** *?thesis* **by** (*simp add*: *int-iffI*)
qed

lemma *OrChopEqvRule*:
assumes $\vdash f = (f1 \vee f2)$
shows $\vdash f;g = ((f1;g) \vee (f2;g))$
proof −
have *1*: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;g = ((f1 \vee f2);g)$ **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash (f1 \vee f2);g = (f1;g \vee f2;g)$ **by** (*rule OrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *NextImpNext*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \bigcirc f \longrightarrow \bigcirc g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ (f \longrightarrow g)$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** (*rule BoxChopImpChop*)
have *4*: $\vdash (skip;f) \longrightarrow (skip;g)$ **by** (*metis 2 3 MP*)

from *4* **show** *?thesis* **by** (*metis next-d-def*)
qed

lemma *ChopOrImpRule*:
assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f;g \longrightarrow (f;g1) \vee (f;g2)$
proof −
have *1*: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;g \longrightarrow f;(g1 \vee g2)$ **by** (*rule RightChopImpChop*)
have *3*: $\vdash f;(g1 \vee g2) = (f;g1 \vee f;g2)$ **by** (*rule ChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *NextImpDist*:
$\vdash \bigcirc (f \longrightarrow g) \longrightarrow \bigcirc f \longrightarrow \bigcirc g$
proof −
have *1*: $\vdash (\neg (f \longrightarrow g)) = (f \wedge \neg g)$ **by** *auto*
hence *2*: $\vdash skip;(\neg (f \longrightarrow g)) = skip;(f \wedge \neg g)$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f \longrightarrow g \vee (f \wedge \neg g)$ **by** *auto*
hence *4*: $\vdash skip;f \longrightarrow (skip;g) \vee (skip;(f \wedge \neg g))$ **by** (*rule ChopOrImpRule*)
hence *5*: $\vdash \neg (skip;(f \wedge \neg g)) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** *auto*
have *6*: $\vdash \neg (skip;(\neg(f \longrightarrow g))) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **using** *2 5* **by** *fastforce*
hence *7*: $\vdash \neg (\bigcirc(\neg(f \longrightarrow g))) \longrightarrow (\bigcirc f) \longrightarrow (\bigcirc g)$ **by** (*simp add*: *next-d-def*)
have *8*: $\vdash \bigcirc(f \longrightarrow g) \longrightarrow \neg (\bigcirc(\neg(f \longrightarrow g)))$ **by** (*rule NextImpNotNextNot*)
from *7 8* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FiniteChopImpDiamond*:
$\vdash (f \wedge finite);g \longrightarrow \Diamond g$
proof −
have *1*: $\vdash f \wedge finite \longrightarrow finite$ **by** *auto*
hence *2*: $\vdash (f \wedge finite);g \longrightarrow finite;g$ **by** (*rule LeftChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *NowImpDiamond*:
$\vdash f \longrightarrow \Diamond f$
proof −
have *1*: $\vdash empty;f = f$ **by** (*rule EmptyChop*)
have *2*: $\vdash empty \longrightarrow finite$ **by** (*rule EmptyImpFinite*)
hence *3*: $\vdash empty;f \longrightarrow finite;f$ **by** (*rule LeftChopImpChop*)
have *4*: $\vdash f \longrightarrow finite;f$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *BoxElim*:
$\vdash \Box f \longrightarrow f$
proof −
have *1*: $\vdash \neg f \longrightarrow \Diamond (\neg f)$ **by** (*rule NowImpDiamond*)

hence *2*: $\vdash \neg\ (\Diamond\ (\neg\ f)) \longrightarrow f$ **by** *auto*
from *2* **show** *?thesis* **by** (*metis always-d-def*)
qed

lemma *NextDiamondImpDiamond*:
$\vdash \bigcirc\ (\Diamond\ f) \longrightarrow \Diamond\ f$
proof −
have *1*: $\vdash$ *skip*;(*finite*;*f*) = ((*skip*;*finite*);*f*) **by** (*rule ChopAssoc*)
hence *2*: $\vdash$ (*skip*;*finite*);*f* = *skip*;(*finite*;*f*) **by** *auto*
hence *3*: $\vdash$ (*skip*;*finite*);*f* = $\bigcirc(\Diamond f)$ **by** (*simp add*: *next-d-def sometimes-d-def*)
have *4*: $\vdash$ (*skip*;*finite*);*f* $\longrightarrow \Diamond\ f$
by (*simp add*: *SkipChopFiniteImpFinite LeftChopImpChop sometimes-d-def*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpNowAndWeakNext*:
$\vdash \Box\ f \longrightarrow (f \wedge$ *wnext* ($\Box\ f$))
proof −
have *1*: $\vdash \neg\ f \longrightarrow \Diamond\ (\neg\ f)$ **by** (*rule NowImpDiamond*)
hence *2*: $\vdash \neg\ (\Diamond\ (\neg\ f)) \longrightarrow f$ **by** *auto*
hence *3*: $\vdash \Box\ f \longrightarrow f$ **by** (*metis always-d-def*)
have *4*: $\vdash \bigcirc$ ($\Diamond\ (\neg\ f)) \longrightarrow \Diamond$ ($\neg\ f$) **by** (*rule NextDiamondImpDiamond*)
have *5*: $\vdash \neg\ \neg\ (\Diamond\ (\neg\ f)) \longrightarrow \Diamond(\ \neg\ f\ )$ **by** *auto*
hence *6*: $\vdash \bigcirc$ ($\neg\ \neg\ (\Diamond\ (\neg\ f))$) $\longrightarrow \bigcirc\ (\Diamond(\ \neg\ f\ ))$ **by** (*rule NextImpNext*)
have *7*: $\vdash \bigcirc$ ($\neg\ \neg\ (\Diamond\ (\neg\ f))$) $\longrightarrow \Diamond$ ($\neg\ f$) **using** *4 6* **by** *auto*
hence *8*: $\vdash \bigcirc$ ($\neg(\ \Box\ f)) \longrightarrow \Diamond$ ($\neg\ f$) **by** (*simp add*: *always-d-def*)
hence *9*: $\vdash \neg\ (\Diamond$ ($\neg\ f$)) $\longrightarrow \neg$ ($\bigcirc$ ($\neg(\ \Box\ f)$))) **by** *auto*
hence *10*:$\vdash \Box f \longrightarrow$ *wnext* ($\Box\ f$) **by** (*simp add*: *always-d-def wnext-d-def*)
from *3 10* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Box\ f \longrightarrow \Box\ g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow\ \neg\ f$ **by** *auto*
hence *3*: $\vdash \Box(\neg\ g \longrightarrow\ \neg\ f)$ **by** (*rule BoxGen*)
have *4*: $\vdash\ \Box(\neg\ g \longrightarrow\ \neg\ f) \longrightarrow$ (*finite*;($\neg g$)) $\longrightarrow$ (*finite*;($\neg f$)) **by** (*rule BoxChopImpChop*)
have *5*: $\vdash$ (*finite*;($\neg g$)) $\longrightarrow$ (*finite*;($\neg f$)) **using** *3 4 MP* **by** *blast*
hence *6*: $\vdash \Diamond\ (\neg g)\ \longrightarrow \Diamond(\neg f)$ **by** (*simp add*: *sometimes-d-def*)
hence *7*: $\vdash\ \neg\ (\Diamond(\neg f)\ ) \longrightarrow \neg(\ \Diamond\ (\neg g))$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *always-d-def*)
qed

lemma *BoxImpDist*:
$\vdash\ \Box(f \longrightarrow g) \longrightarrow \Box\ f \longrightarrow \Box\ g$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg\ \ g \longrightarrow \neg\ \ f)$
by *auto*

hence *2*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Box(\neg\ g \longrightarrow \neg\ f)$
by (*rule BoxImpBoxRule*)
have *3*: $\vdash \Box((\neg\ g) \longrightarrow \neg\ f) \longrightarrow (\mathit{finite}; (\neg\ g)) \longrightarrow (\mathit{finite}; (\neg\ f))$
by (*rule BoxChopImpChop*)
have *4*: $\vdash \Box(f \longrightarrow g) \longrightarrow (\mathit{finite}; (\neg\ g)) \longrightarrow (\mathit{finite}; (\neg\ f))$
using *2 3 lift-imp-trans* **by** *blast*
hence *5*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Diamond(\neg\ g) \longrightarrow \Diamond(\neg\ f)$
by (*simp add*: *sometimes-d-def*)
hence *6*: $\vdash \Box(f \longrightarrow g) \longrightarrow \neg(\Diamond(\neg\ f)) \longrightarrow \neg(\Diamond(\neg\ g))$
by *auto*
from *6* **show** *?thesis* **by** (*simp add*: *always-d-def*)
qed

lemma *DiamondEmptyEqvFinite*:
$\vdash \Diamond\ \mathit{empty} = \mathit{finite}$
proof −
have *1*: $\vdash \mathit{finite};\ \mathit{empty} = \mathit{finite}$ **by** (*rule ChopEmpty*)
from *1* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *NextEqvNext*:
assumes $\vdash f = g$
shows $\vdash \bigcirc f = \bigcirc g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \mathit{skip};f = \mathit{skip};g$ **by** (*rule RightChopEqvChop*)
from *1* **show** *?thesis* **by** (*metis 2 next-d-def*)
qed

lemma *NextAndNextImpNextRule*:
assumes $\vdash (f \wedge g) \longrightarrow h$
shows $\vdash (\bigcirc f \wedge \bigcirc g) \longrightarrow \bigcirc h$
using *assms*
by (*simp add*: *Valid-def itl-defs*)

lemma *NextAndNextEqvNextRule*:
assumes $\vdash (f \wedge g) = h$
shows $\vdash (\bigcirc f \wedge \bigcirc g) = \bigcirc h$
using *assms*
by (*simp add*: *NextAndNextImpNextRule NextImpNext Prop11 Prop12*)

lemma *WeakNextEqvWeakNext*:
assumes $\vdash f = g$
shows $\vdash \mathit{wnext}\ f = \mathit{wnext}\ g$
using *assms* **using** *inteq-reflection* **by** *force*

lemma *DiamondImpDiamond*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Diamond f \longrightarrow \Diamond g$
using *assms* **by** (*simp add*: *RightChopImpChop sometimes-d-def*)

lemma *DiamondEqvDiamond*:
assumes $\vdash f = g$
shows $\vdash \Diamond f = \Diamond g$
using *assms* **using** *int-eq* **by** *force*

lemma *BoxEqvBox*:
assumes $\vdash f = g$
shows $\vdash \Box f = \Box g$
using *assms* **using** *inteq-reflection* **by** *force*

lemma *BoxAndBoxImpBoxRule*:
assumes $\vdash f \wedge g \longrightarrow h$
shows $\vdash \Box f \wedge \Box g \longrightarrow \Box h$
using *assms* **by** (*auto simp*: *itl-defs Valid-def*)

lemma *BoxAndBoxEqvBoxRule*:
assumes $\vdash (f \wedge g) = h$
shows $\vdash (\Box f \wedge \Box g) = \Box h$
using *assms BoxAndBoxImpBoxRule BoxImpBoxRule* **by** (*metis int-iffD1 int-iffD2 int-iffI Prop12*)

lemma *ImpBoxRule*:
assumes $\vdash \quad f \longrightarrow g$
shows $\vdash \Box f \longrightarrow \Box g$
using *assms* **by** (*simp add*: *BoxImpBoxRule*)

lemma *WnextEqvEmptyOrNext*:
$\vdash$ *wnext* $f = (empty \vee \bigcirc f)$
by (*auto simp*: *Valid-def itl-defs zero-enat-def*)

lemma *BoxIntro*:
assumes $\vdash \quad f \longrightarrow g$
$\vdash \quad more \ \wedge f \longrightarrow \bigcirc f$
shows $\vdash f \longrightarrow \Box g$
proof −
have *1*: $\vdash \quad more \ \wedge f \longrightarrow \bigcirc f$
using *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow (empty \vee \bigcirc f)$
unfolding *empty-d-def* **by** *fastforce*
hence *3*: $\vdash f \longrightarrow$ *wnext* f
by (*metis WnextEqvEmptyOrNext inteq-reflection*)
hence *4*: $\vdash \Box(f \longrightarrow$ *wnext* $f)$
by (*rule BoxGen*)
have *5*: $\vdash (\Box\ (f \longrightarrow$ *wnext* $f)) \wedge f \longrightarrow \Box f$
by (*rule BoxInduct*)
hence *6*: $\vdash (\Box\ (f \longrightarrow$ *wnext* $f)) \longrightarrow (f \longrightarrow \Box f)$
by *fastforce*
have *7*: $\vdash f \longrightarrow \Box f$
using *4 6 MP* **by** *blast*
have *8*: $\vdash \Box f \longrightarrow f$

by (*rule BoxElim*)
have *9*: $\vdash f = \Box f$
using *7 8* **by** *fastforce*
have *10*: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
hence *11*: $\vdash \Box f \longrightarrow \Box\ g$
by (*rule ImpBoxRule*)
from *7 9 11* **show** *?thesis*
using *lift-imp-trans* **by** *blast*
qed

lemma *NextLoop*:
assumes $\vdash f \longrightarrow \bigcirc f$
shows $\vdash$ *finite* $\longrightarrow \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow \bigcirc f$
using *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow (more \wedge wnext\ f)$
unfolding *more-d-def wnext-d-def*
by (*metis NextImpNext NextImpNotNextNot Prop05 Prop10 Prop12 int-iffD1 int-simps*(*29*) *lift-imp-trans*)
hence *3*: $\vdash f \longrightarrow wnext\ f$
by *auto*
hence *4*: $\vdash \Box(f \longrightarrow wnext\ f)$
by (*rule BoxGen*)
have *5*: $\vdash \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f$
by (*rule BoxInduct*)
hence *6*: $\vdash \Box\ (f \longrightarrow wnext\ f) \longrightarrow (f \longrightarrow \Box f)$
by *fastforce*
have *7*: $\vdash f \longrightarrow \Box f$
using *4 6 MP* **by** *blast*
have *8*: $\vdash \Box f \longrightarrow f$
by (*rule BoxElim*)
have *9*: $\vdash f = \Box f$
using *7 8* **by** *fastforce*
have *10*: $\vdash f \longrightarrow more$
using *2* **by** *auto*
hence *11*: $\vdash \Box\ f \longrightarrow \Box\ more$
by (*rule ImpBoxRule*)
have *12*: $\vdash$ *finite* $= (\neg(\Box\ more))$
by (*metis DiamondEmptyEqvFinite InfEqvNotFinite always-d-def empty-d-def finite-d-def int-eq*)
from *7 9 11 12* **show** *?thesis*
by *fastforce*
qed

lemma *NotEmptyAndNext*:
$\vdash \neg(empty \wedge \bigcirc f)$
by (*auto simp*: *Valid-def itl-defs zero-enat-def*)

lemma *BoxEqvAndWnextBox*:
$\vdash \Box\ f = (f \wedge wnext\ (\ \Box\ f))$

proof −
have *1*: $\vdash \Box\ f \longrightarrow f \wedge$ *wnext* $(\ \Box\ f)$
using *BoxImpNowAndWeakNext* **by** *blast*
have *2*: $\vdash f \wedge$ *wnext* $(\ \Box\ f) \longrightarrow f$
by *auto*
have *3*: $\vdash$ *more* $\wedge\ (f \wedge$ *wnext* $(\ \Box\ f)\ ) \longrightarrow \bigcirc\ (f \wedge$ *wnext* $(\ \Box\ f)\ )$
using *1 NextImpNext WnextEqvEmptyOrNext empty-d-def int-iffD1*
by (*metis Prop01 Prop05 Prop08*)
have *4*: $\vdash f \wedge$ *wnext* $(\ \Box\ f) \longrightarrow \Box\ f$
using *2 3 BoxIntro* **by** *blast*
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxEqvAndEmptyOrNextBox*:
$\vdash \Box f = (f \wedge (empty\ \vee\ \bigcirc(\Box\ f)))$
using *BoxEqvAndWnextBox WnextEqvEmptyOrNext* **by** (*metis int-eq*)

lemma *BoxEqvBoxBox*:
$\vdash \Box f = \Box\ (\Box\ f)$
using *BoxGen BoxInduct*
by (*metis BoxImpNowAndWeakNext MP int-iffI Prop09 Prop12*)

lemma *BoxBoxImpBox*:
$\vdash \Box(\Box h) \longrightarrow \Box\ h$
by (*simp add*: *BoxElim*)

lemma *BoxImpBoxBox*:
$\vdash \Box\ h\ \longrightarrow\ \Box(\Box h)$
by (*simp add*: *BoxEqvBoxBox int-iffD1*)

lemma *DiamondIntroC*:
assumes $\vdash f \longrightarrow \bigcirc\ g$
shows $\vdash f \longrightarrow \Diamond\ g$
using *assms*
by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopSkipEqvSkipChopFinite NextChop NextDiamondImpDiamond NowImpDiamond inteq-reflection lift-imp-trans next-d-def sometimes-d-def*)

lemma *DiamondIntro*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow \bigcirc f$
shows $\vdash f \wedge$ *finite* $\longrightarrow \Diamond\ g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow \bigcirc f$
using *assms* **by** *auto*
hence *2*: $\vdash f \wedge \neg\ g \wedge (\Box\ (\neg\ g)) \longrightarrow (\bigcirc f) \wedge (\Box\ (\neg\ g))$
by *auto*
have *3*: $\vdash (\Box\ (\neg\ g)) \longrightarrow \neg\ g$
by (*rule BoxElim*)
hence *4*: $\vdash \Box\ (\neg\ g) = ((\Box\ (\neg\ g)) \wedge \neg\ g)$
using *BoxImpBoxBox BoxBoxImpBox* **by** *fastforce*

have *5*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \bigcirc\ f \wedge \Box\ (\neg\ g)$
using *2 4* **by** *fastforce*
have *6*: $\vdash \Box\ (\neg\ g) = ((\neg\ g) \wedge wnext(\Box\ (\neg\ g)))$
using *BoxEqvAndWnextBox* **by** *metis*
have *7*: $\vdash \bigcirc\ f \wedge \Box\ (\neg\ g) \longrightarrow\ \bigcirc\ f \wedge wnext(\Box\ (\neg\ g))$
using *6* **by** *auto*
have *8*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \bigcirc\ f \wedge wnext(\Box\ (\neg\ g))$
using *5 7* **using** *lift-imp-trans* **by** *blast*
hence *9*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow more \wedge wnext\ f \wedge wnext(\Box\ (\neg\ g))$
using *zero-enat-def* **by** (*auto simp*: *Valid-def itl-defs*)
hence *10*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ f \wedge wnext(\Box\ (\neg\ g))$
by *auto*
hence *11*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box\ (\neg\ g))$
by (*auto simp*: *Valid-def itl-defs*)
hence *12*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box\ (\neg\ g)))$
by (*rule BoxGen*)
have *13*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box\ (\neg\ g))) \wedge f \wedge (\Box\ (\neg\ g)) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g)))$
by (*rule BoxInduct*)
hence *14*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box(\neg\ g))) \longrightarrow ((f \wedge (\Box(\neg\ g))) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g))))$
by *fastforce*
have *15*: $\vdash ((f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g))))$
using *12 14 MP* **by** *blast*
have *16*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) \longrightarrow (f \wedge (\Box\ (\neg\ g)))$
by (*rule BoxElim*)
have *17*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) = (f \wedge (\Box\ (\neg\ g)))$
using *16 15* **by** *fastforce*
have *18*: $\vdash (f \wedge (\Box\ (\neg\ g))) \longrightarrow more$
using *9* **by** *auto*
hence *19*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box\ more$
by (*rule ImpBoxRule*)
have *20*: $\vdash finite = (\neg(\Box\ more))$
by (*metis DiamondEmptyEqvFinite InfEqvNotFinite always-d-def empty-d-def finite-d-def int-eq*)
have *21*: $\vdash finite \longrightarrow \neg(f \wedge (\Box\ (\neg\ g)))$
using *17 19 20* **by** *fastforce*
hence *22*: $\vdash finite \longrightarrow \neg\ f \vee \neg\ (\Box\ (\neg\ g))$
by *auto*
have *23*: $\vdash (\neg\ (\Box\ (\neg\ g))) = \Diamond\ g$
by (*auto simp*: *always-d-def*)
from *22 23* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondIntroB*:
assumes $\vdash (f \wedge \neg\ \ g) \longrightarrow \bigcirc\ (f \wedge \neg\ g)$
shows $\vdash f \wedge finite \longrightarrow \Diamond\ g$
proof −
have *1*: $\vdash (f \wedge \neg\ \ g) \longrightarrow \bigcirc\ (f \wedge \neg g)$ **using** *assms* **by** *auto*
hence *2*: $\vdash finite \longrightarrow \neg(f \wedge \neg\ g)$ **by** (*rule NextLoop*)
hence *3*: $\vdash f \wedge finite \longrightarrow g$ **by** *auto*
have *4*: $\vdash g \longrightarrow \Diamond\ g$ **by** (*rule NowImpDiamond*)
from *3 4* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*

qed

lemma *NextContra* :
assumes $\vdash (f \wedge \neg\ g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$
shows $\vdash f \wedge \mathit{finite} \longrightarrow g$
proof −
have *1*: $\vdash (f \wedge \neg\ g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*auto simp*: *itl-defs Valid-def*)
hence *3*: $\vdash \mathit{finite} \longrightarrow \neg\neg(f \longrightarrow g)$ **by** (*rule NextLoop*)
from *3* **show** *?thesis* **by** *auto*
qed

lemma *DiamondDiamondEqvDiamond*:
$\vdash \Diamond(\Diamond f) = \Diamond f$
proof −
have *1*: $\vdash \mathit{finite};\mathit{finite} = \mathit{finite}$ **by** (*simp add*: *FiniteChopFiniteEqvFinite*)
hence *2*: $\vdash (\mathit{finite};\mathit{finite});f = \mathit{finite};f$ **using** *LeftChopEqvChop* **by** *blast*
have *3*: $\vdash (\mathit{finite};\mathit{finite});f = \mathit{finite};(\mathit{finite};f)$ **using** *ChopAssoc* **by** *fastforce*
from *2 3* **show** *?thesis* **by** (*metis inteq-reflection sometimes-d-def*)
qed

lemma *WeakNextDiamondInduct*:
assumes $\vdash \mathit{wnext}\ (\Diamond f) \longrightarrow f$
shows $\vdash \mathit{finite} \longrightarrow f$
proof −
have *1*: $\vdash \mathit{wnext}\ (\Diamond f) \longrightarrow f$ **using** *assms* **by** *blast*
hence *2*: $\vdash \neg f \longrightarrow \neg(\mathit{wnext}\ (\Diamond f))$ **by** *fastforce*
hence *3*: $\vdash \neg f \longrightarrow \bigcirc(\neg(\Diamond f))$ **by** (*simp add*: *wnext-d-def*)
have *4*: $\vdash f \longrightarrow \Diamond f$ **by** (*rule NowImpDiamond*)
hence *5*: $\vdash \neg(\Diamond f) \longrightarrow \neg f$ **by** *auto*
have *6*: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **using** *3 5* **using** *NextImpNext lift-imp-trans* **by** *blast*
hence *7*: $\vdash \mathit{finite} \longrightarrow \neg\neg f$ **by** (*rule NextLoop*)
from *7* **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInducta*:
assumes $\vdash \mathit{empty} \longrightarrow f$
$\vdash \bigcirc f \longrightarrow f$
shows $\vdash \mathit{finite} \longrightarrow f$
proof −
have *1*: $\vdash \mathit{empty} \longrightarrow f$ **using** *assms* **by** *auto*
have *2*: $\vdash \bigcirc f \longrightarrow f$ **using** *assms* **by** *blast*
have *3*: $\vdash (\mathit{empty} \vee \bigcirc f) \longrightarrow f$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash \mathit{wnext}\ f = (\mathit{empty} \vee \bigcirc f)$ **by** (*rule WnextEqvEmptyOrNext*)
hence *5*: $\vdash \mathit{wnext}\ f \longrightarrow f$ **using** *3* **by** *fastforce*
hence *6*: $\vdash \neg f \longrightarrow \neg(\mathit{wnext}\ f)$ **by** *auto*
hence *7*: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **by** (*auto simp*: *wnext-d-def*)
hence *8*: $\vdash \mathit{finite} \longrightarrow \neg\neg f$ **by** (*rule NextLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInductb*:
assumes $\vdash$ *empty* $\wedge f \longrightarrow g$
$\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$
shows $\vdash f \wedge$ *finite* $\longrightarrow g$
proof −
have *1*: $\vdash$ *empty* $\wedge f \longrightarrow g$ **using** *assms* **by** *auto*
have *2*: $\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *assms* **by** *blast*
have *3*: $\vdash$ (*empty* $\vee \bigcirc(f \longrightarrow g)) \wedge f \longrightarrow g$ **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *wnext* $(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*
hence *5*: $\vdash$ *wnext* $(f \longrightarrow g) \longrightarrow (f \longrightarrow g)$ **by** *fastforce*
hence *6*: $\vdash \neg (f \longrightarrow g) \longrightarrow \neg$ (*wnext* $(f \longrightarrow g))$ **by** *fastforce*
hence *7*: $\vdash \neg (f \longrightarrow g) \longrightarrow \bigcirc (\neg(f \longrightarrow g))$ **by** (*simp add*: *wnext-d-def*)
hence *8*: $\vdash$ *finite* $\longrightarrow \neg \neg (f \longrightarrow g)$ **by** (*rule NextLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *FinImpFin*:
assumes $\vdash f \longrightarrow g$
shows $\vdash$ *fin* $f \longrightarrow$ *fin* g
using *ImpBoxRule*[*of LIFT* (*empty* $\longrightarrow f$) *LIFT* (*empty* $\longrightarrow g$)] *assms*
fin-d-def[*of f*] *fin-d-def*[*of g*] **by** *fastforce*

lemma *FinEqvFin*:
assumes $\vdash f = g$
shows $\vdash$ *fin* $f =$ *fin* g
using *assms* **by** (*simp add*: *FinImpFin Prop11*)

lemma *FinAndFinImpFinRule*:
assumes $\vdash f \wedge g \longrightarrow h$
shows $\vdash$ *fin* $f \wedge$ *fin* $g \longrightarrow$ *fin* h
using *assms* **by** (*auto simp add*: *itl-defs Valid-def*)

lemma *FinAndFinEqvFinRule*:
assumes $\vdash (f \wedge g) = h$
shows $\vdash$ (*fin* $f \wedge$ *fin* $g) =$ *fin* h
using *assms*
by (*simp add*: *FinAndFinImpFinRule FinImpFin Prop11 Prop12*)

lemma *HaltEqvHalt*:
assumes $\vdash f = g$
shows $\vdash$ *halt* $f =$ *halt* g
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*empty* $= f) = (empty = g)$ **by** *auto*
hence *3*: $\vdash \Box(empty = f) = \Box\ (empty = g)$ **by** (*rule BoxEqvBox*)
from *3* **show** *?thesis* **by** (*simp add*: *halt-d-def*)
qed

lemma *BiImpDiImpDi*:
$\vdash bi\ (f \longrightarrow g) \longrightarrow\ di\ \ f \longrightarrow\ \ di\ \ g$
proof −
have *1*: $\vdash bi\ (f \longrightarrow g)\ \longrightarrow (f;\ \#\mathit{True}) \longrightarrow (g;\ \#\mathit{True})$ **by** (*rule BiChopImpChop*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiImpDi*:
assumes $\vdash f \longrightarrow g$
shows $\vdash\ di\ \ f \longrightarrow\ \ di\ \ g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ \#\mathit{True} \longrightarrow g;\ \#\mathit{True}$ **by** (*rule LeftChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *BiImpBiRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash bi\ f \longrightarrow bi\ g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ \ g \longrightarrow \neg\ \ f$ **by** *auto*
hence *3*: $\vdash\ \ di\ (\neg\ \ g) \longrightarrow\ \ di\ (\neg\ \ f)$ **by** (*rule DiImpDi*)
hence *4*: $\vdash \neg\ (\ di\ (\neg\ \ f)) \longrightarrow \neg\ (\ di\ (\neg\ \ g))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *DiEqvDi*:
assumes $\vdash f = g$
shows $\vdash\ \ di\ f =\ \ di\ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ \#\mathit{True} = g;\ \#\mathit{True}$ **by** (*rule LeftChopEqvChop*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *BiEqvBi*:
assumes $\vdash f = g$
shows $\vdash\ \ bi\ f =\ \ bi\ \ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ \ f) = (\neg\ \ g)$ **by** *auto*
hence *3*: $\vdash\ \ di\ (\neg\ \ f) =\ \ di\ (\neg\ \ g)$ **by** (*rule DiEqvDi*)
hence *4*: $\vdash (\neg\ \ (di\ (\neg\ \ f))) = (\neg\ (\ di\ (\neg\ \ g)))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *LeftChopChopImpChopRule*:
assumes $\vdash (f;\ g) \longrightarrow g$
shows $\vdash (f;\ g);\ h \longrightarrow (g;\ h)$

proof −
have *1*: ⊢ (*f*; *g*) ⟶ *g* **using** *assms* **by** *blast*
hence *2*: ⊢ (*f*; *g*); *h* ⟶ *g*; *h* **by** (*rule LeftChopImpChop*)
have *3*: ⊢ *f*; (*g*; *h*) = (*f*; *g*); *h* **by** (*rule ChopAssoc*)
from *2 3* **show** *?thesis* **by** *auto*
qed

lemma *AndChopCommute* :
⊢ (*f* ∧ *f1*); *g* = (*f1* ∧ *f*); *g*
proof −
have *1*: ⊢ (*f* ∧ *f1*) = (*f1* ∧ *f*) **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftChopEqvChop*)
qed

lemma *BiAndChopImport*:
⊢ *bi f* ∧ (*f1*; *g*) ⟶ (*f* ∧ *f1*);*g*
proof −
have *1*: ⊢ *f* ⟶ (*f1* ⟶ *f*∧ *f1*) **by** *auto*
hence *2*: ⊢ *bi f* ⟶ *bi* (*f1* ⟶ *f*∧ *f1*) **by** (*rule BiImpBiRule*)
have *3*: ⊢ *bi* (*f1* ⟶ (*f* ∧ *f1*)) ⟶ *f1*; *g* ⟶ (*f* ∧ *f1*); *g* **by** (*rule BiChopImpChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *StateAndChopImport*:
⊢ (*init w*) ∧ (*f*;*g*) ⟶ ((*init w*) ∧ *f*);*g*
proof −
have *1*: ⊢ (*init w*)⟶ *bi* (*init w*) **by** (*rule StateImpBi*)
hence *2*: ⊢ (*init w*) ∧ (*f*; *g*) ⟶ *bi* (*init w*) ∧ (*f*; *g*) **by** *auto*
have *3*: ⊢ *bi* (*init w*) ∧ (*f*; *g*) ⟶ ((*init w*) ∧ *f*); *g* **by** (*rule BiAndChopImport*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

3.3.5 Further Properties Di and Bi

lemma *ImpDi*:
⊢ *f* ⟶ *di f*
proof −
have *1*: ⊢ *f*; *empty* = *f* **by** (*rule ChopEmpty*)
have *2*: ⊢ *empty* ⟶ #*True* **by** *auto*
hence *3*: ⊢ *f*; *empty* ⟶ *f*; #*True* **by** (*rule RightChopImpChop*)
have *4* : ⊢ *f* ⟶ *f*; #*True* **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiState*:
⊢ *di* (*init w*) = (*init w*)
proof −
have *0*: ⊢ (*init* (¬*w*)) ⟶ *bi* (*init* (¬*w*)) **using** *StateImpBi* **by** *fastforce*
hence *1*: ⊢ ¬(*init w*) ⟶ *bi* (¬ (*init w*)) **using** *Initprop*(*2*) **by** (*metis inteq-reflection*)
hence *2*: ⊢ (¬ (*init w*)) ⟶ ¬ (*di* (¬ ¬ (*init w*))) **by** (*simp add*: *bi-d-def*)

have *3*: $\vdash (\neg\ (init\ w) \longrightarrow \neg\ (di\ (\neg\ \neg\ (init\ w)))) \longrightarrow$
$(\ di\ (\neg\ \neg\ (init\ w)) \longrightarrow (init\ w))$ **by** *auto*
have *4*: $\vdash\ di\ (\neg\ \neg\ (init\ w)) \longrightarrow (init\ w)$ **using** *2 3 MP* **by** *blast*
have *5*: $\vdash (init\ w) \longrightarrow \neg\ \neg\ (init\ w)$ **by** *auto*
hence *6*: $\vdash\ di\ (init\ w) \longrightarrow\ di\ (\neg\ \neg\ (init\ w))$ **by** (*rule DiImpDi*)
have *7*: $\vdash\ di\ (init\ w) \longrightarrow (init\ w)$ **using** *6 4* **using** *lift-imp-trans* **by** *metis*
have *8*: $\vdash (init\ w) \longrightarrow\ di\ (init\ w)$ **by** (*rule ImpDi*)
from *7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *StateChop*:
$\vdash (init\ w);\ f \longrightarrow (init\ w)$
by (*metis ChopAssoc Prop12 RightChopImpChop TrueW init-d-def int-simps*(*13*) *int-simps*(*17*)
inteq-reflection)

lemma *StateChopExportA*:
$\vdash ((init\ w) \wedge f);\ g \longrightarrow (init\ w)$
using *DiState AndChopA StateChop* **by** *fastforce*

lemma *StateAndChop*:
$\vdash ((init\ w) \wedge f);\ g = ((init\ w) \wedge (f;\ g))$
by (*simp add*: *AndChopB StateAndChopImport StateChopExportA Prop11 Prop12*)

lemma *StateAndChopImpChopRule*:
assumes $\vdash (init\ w) \wedge f \longrightarrow f1$
shows $\vdash (init\ w) \wedge (f;\ g) \longrightarrow (f1;\ g)$
proof −
have *1*: $\vdash (init\ w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash ((init\ w) \wedge f);\ g \longrightarrow f1;\ g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash ((init\ w) \wedge f);\ g = ((init\ w) \wedge (f;\ g))$ **by** (*rule StateAndChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpChopEqvChop* :
assumes $\vdash (init\ w) \longrightarrow (f = f1)$
shows $\vdash (init\ w) \longrightarrow ((f;\ g) = (f1;\ g))$
proof −
have *1*: $\vdash (init\ w) \longrightarrow (f = f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash (init\ w) \wedge f \longrightarrow f1$ **by** *auto*
hence *3*: $\vdash (init\ w) \wedge (f;\ g) \longrightarrow (f1;\ g)$ **by** (*rule StateAndChopImpChopRule*)
have *4*: $\vdash (init\ w) \wedge f1 \longrightarrow f$ **using** *1* **by** *auto*
hence *5*: $\vdash (init\ w) \wedge (f1;\ g) \longrightarrow (f;\ g)$ **by** (*rule StateAndChopImpChopRule*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvStateAndChop*:
assumes $\vdash f = (init\ w) \wedge f1$
shows $\vdash (f;\ g) = ((init\ w) \wedge (f1;\ g))$
proof −
have *1*: $\vdash f = ((init\ w) \wedge f1)$ **using** *assms* **by** *auto*

hence *2*: $\vdash f;\ g = (((init\ w) \wedge f1);\ g)$ **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash ((init\ w) \wedge f1);\ g = ((init\ w) \wedge (f1;\ g))$ **by** (*rule StateAndChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DiIntro*:
$\vdash f \longrightarrow\ di\ \ f$
proof −
have *1*: $\vdash f;empty\ = f$ **by** (*rule ChopEmpty*)
have *2*: $\vdash\ \ empty\ \longrightarrow \#True$ **by** *auto*
hence *3*: $\vdash \Box(\ empty\ \longrightarrow \#True)$ **by** (*rule BoxGen*)
have *4*: $\vdash \Box(\ empty\ \longrightarrow \#True) \longrightarrow\ (f;\ \ empty\ \longrightarrow f;\ \#True)$ **by** (*rule BoxChopImpChop*)
have *5*: $\vdash f;empty\ \longrightarrow f;\#True$ **using** *3 4 MP* **by** *fastforce*
hence *6*: $\vdash f;empty\ \longrightarrow\ \ di\ \ f$ **by** (*simp add*: *di-d-def*)
from *1 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BiElim*:
$\vdash\ \ \ bi\ \ f \longrightarrow f$
proof −
have *1*: $\vdash \neg f \longrightarrow\ \ di\ (\neg f)$ **by** (*rule DiIntro*)
have *2*: $\vdash (\neg f \longrightarrow\ \ di\ (\neg f)) \longrightarrow (\neg(di\ (\neg f)) \longrightarrow f)$ **by** *auto*
have *3*: $\vdash \neg\ \ (di\ (\neg f)) \longrightarrow f$ **using** *1 2 MP* **by** *blast*
from *3* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *BiContraPosImpDist*:
$\vdash bi\ (\neg g \longrightarrow \neg f) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$
proof −
have *1*: $\vdash bi\ (\neg g \longrightarrow \neg f) \longrightarrow (di\ (\neg g)) \longrightarrow (di\ (\neg f))$ **by** (*rule BiImpDiImpDi*)
hence *2*: $\vdash bi\ (\neg g \longrightarrow \neg f) \longrightarrow (\neg(\ di\ (\neg f))) \longrightarrow (\neg(di\ (\neg g)))$ **by** *auto*
from *2* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *BiImpDist*:
$\vdash bi\ (f \longrightarrow g) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg\ \ g \longrightarrow \neg\ \ f)$ **by** *auto*
hence *2*: $\vdash \neg\ (\neg\ \ g \longrightarrow \neg\ \ f) \longrightarrow \neg\ (f \longrightarrow g)$ **by** *auto*
hence *3*: $\vdash bi\ (\neg\ (\neg\ \ g \longrightarrow \neg\ \ f) \longrightarrow \neg\ (f \longrightarrow g))$ **by** (*rule BiGen*)
have *4*: $\vdash bi\ (\neg\ (\neg\ \ g \longrightarrow \neg\ \ f) \longrightarrow \neg\ (f \longrightarrow g))$
$\longrightarrow$
$bi\ (f \longrightarrow g) \longrightarrow bi\ (\neg\ \ g \longrightarrow \neg\ \ f)$ **by** (*rule BiContraPosImpDist*)
have *5*: $\vdash bi\ (f \longrightarrow g) \longrightarrow bi\ (\neg\ \ g \longrightarrow \neg\ \ f)$ **using** *3 4 MP* **by** *blast*
have *6*: $\vdash bi\ (\neg\ \ g \longrightarrow \neg\ \ f) \longrightarrow (bi\ \ f) \longrightarrow (bi\ \ g)$ **by** (*rule BiContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *IfChopEqvRule*:
assumes $\vdash f =\ \ if_i\ \ (init\ w)\ \ then\ \ f1\ \ else\ \ f2$

shows $\vdash f;\ g = \ if_i\ \ (init\ w)\ \ then\ \ (f1;\ g)\ \ else\ \ (f2;\ g)$
proof −
have *1*: $\vdash f = \ if_i\ \ (init\ w)\ \ then\ \ f1\ \ else\ \ f2$
using *assms* **by** *auto*
hence *2*: $\vdash f = (((init\ w) \wedge f1) \vee\ \ (\ (init\ (\neg\ w)) \wedge f2))$
by (*metis Initprop*(*2*) *ifthenelse-d-def inteq-reflection*)
hence *3*: $\vdash f;\ g = (((init\ w) \wedge f1);\ g \vee\ \ (\ \ (init\ (\neg\ w)) \wedge f2);\ g)$
by (*rule OrChopEqvRule*)
have *4*: $\vdash ((init\ w) \wedge f1);\ g = ((init\ w) \wedge (f1;\ g))$
by (*rule StateAndChop*)
have *5*: $\vdash (\ \ (init\ (\neg\ w)) \wedge f2);\ g = \ \ ((init\ (\neg\ w)) \wedge (f2;\ g))$
by (*rule StateAndChop*)
have *6*: $\vdash f;\ g = (((init\ w) \wedge f1;\ g)\ \ \vee\ \ (\ \ (init\ (\neg\ w)) \wedge f2;\ g))$
using *3 4 5* **by** *fastforce*
from *6* **show** *?thesis*
by (*metis Initprop*(*2*) *ifthenelse-d-def inteq-reflection*)
qed

lemma *ChopOrEqvRule*:
assumes $\vdash g = (g1 \vee\ \ g2)$
shows $\vdash f;g = ((f;g1) \vee\ \ (f;g2))$
proof −
have *1*: $\vdash g = (g1 \vee\ \ g2)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ g = (f;\ (g1 \vee\ \ g2))$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f;\ (g1 \vee\ \ g2) = (f;\ g1 \vee\ \ f;\ g2)$ **by** (*rule ChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrChopEqv*:
$\vdash (\ empty\ \ \vee\ \ f);\ g = (g \vee\ \ (f;\ g))$
proof −
have *1*: $\vdash (\ empty\ \ \vee\ \ f);\ g = ((\ empty\ ;\ g) \vee\ \ (f;\ g))$ **by** (*rule OrChopEqv*)
have *2*: $\vdash\ \ empty\ ;\ g = g$ **by** (*rule EmptyChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopEqv*:
$\vdash (\ empty\ \ \vee\ \ \bigcirc f);\ g = (g \vee\ \ \bigcirc(f;\ g))$
proof −
have *1*: $\vdash (\ empty\ \ \vee\ \ \bigcirc f);\ g = (g \vee\ \ ((\bigcirc f);\ g))$ **by** (*rule EmptyOrChopEqv*)
have *2*: $\vdash (\bigcirc f);\ g = \bigcirc(f;\ g)$ **by** (*rule NextChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrChopImpRule*:
assumes $\vdash\ \ f \longrightarrow\ \ empty\ \ \vee\ \ f1$
shows $\vdash f;\ g \longrightarrow g \vee\ \ (f1;\ g)$
proof −
have *1*: $\vdash f \longrightarrow\ \ empty\ \ \vee\ \ f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ g \longrightarrow (\ empty\ \ \vee\ \ f1);\ g$ **by** (*rule LeftChopImpChop*)

have *3*: $\vdash$ (*empty* $\vee$ *f1*); *g* = (*g* $\vee$ (*f1*; *g*)) **by** (*rule EmptyOrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrChopEqvRule*:
assumes $\vdash$ *f* = (*empty* $\vee$ *f1*)
shows $\vdash$ *f*; *g* = (*g* $\vee$ (*f1*; *g*))
proof −
have *1*: $\vdash$ *f* = (*empty* $\vee$ *f1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* = ((*empty* $\vee$ *f1*); *g*) **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash$ (*empty* $\vee$ *f1*); *g* = (*g* $\vee$ (*f1*; *g*)) **by** (*rule EmptyOrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopImpRule*:
assumes $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ ○ *f1*
shows $\vdash$ *f*; *g* $\longrightarrow$ *g* $\vee$ ○(*f1*; *g*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ ○ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ (*empty* $\vee$ ○ *f1*); *g* **by** (*rule LeftChopImpChop*)
have *3*: $\vdash$ (*empty* $\vee$ ○ *f1*); *g* = (*g* $\vee$ ○(*f1*; *g*)) **by** (*rule EmptyOrNextChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopEqvRule*:
assumes $\vdash$ *f* = (*empty* $\vee$ ○ *f1*)
shows $\vdash$ *f*; *g* = (*g* $\vee$ ○(*f1*; *g*))
proof −
have *1*: $\vdash$ *f* = (*empty* $\vee$ ○ *f1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* = ((*empty* $\vee$ ○ *f1*); *g*) **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash$ (*empty* $\vee$ ○ *f1*); *g* = (*g* $\vee$ ○(*f1*; *g*)) **by** (*rule EmptyOrNextChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEmptyOrImpRule*:
assumes $\vdash$ *g* $\longrightarrow$ *empty* $\vee$ *g1*
shows $\vdash$ *f*; *g* $\longrightarrow$ *f* $\vee$ (*f*; *g1*)
proof −
have *1*: $\vdash$ *g* $\longrightarrow$ *empty* $\vee$ *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ (*f*; *empty*) $\vee$ (*f*; *g1*) **by** (*rule ChopOrImpRule*)
have *3*: $\vdash$ *f*; *empty* = *f* **by** (*rule ChopEmpty*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndEmptyImpBoxState*:
$\vdash$ (*init w*) $\wedge$ *empty* $\longrightarrow$ □ (*init w*)
using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*

lemma *BoxEqvAndBox*:
$\vdash$ □ *f* = (*f* $\wedge$ □ *f*)

using *BoxElim* **by** *fastforce*

lemma *NotBoxImpNotOrNotNextBox*:
$\vdash \neg(\ \Box\ f) \longrightarrow \neg f \vee\ \neg(\ \bigcirc\ (\Box\ f)\ )$
proof −
have *1*: $\vdash f \wedge (\bigcirc\ (\Box\ f)) \longrightarrow \Box\ f$ **using** *BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence *2*: $\vdash \neg(\ \Box\ f) \longrightarrow \neg(f \wedge (\bigcirc\ (\Box\ f))\ )$ **by** *fastforce*
have *3*: $\vdash (\neg(f \wedge (\bigcirc\ (\Box\ f))\ )) = (\neg f \vee \neg(\ \bigcirc\ (\Box\ f)\ )\quad)$ **by** *auto*
from *2 3* **show** *?thesis* **by** *auto*
qed

lemma *BoxStateChopBoxAndInfImpBox*:
$\vdash \Box\ (init\ w);\ \Box\ (init\ w) \wedge inf \longrightarrow \Box\ (init\ w)$
by (*auto simp add*: *Valid-def itl-defs nfirst-eq-nnth-zero*)
(*metis enat-ile le-add-diff-inverse linorder-le-cases min-def ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD ntaken-nnth*)

lemma *BoxStateChopBoxEqvBox*:
$\vdash\quad \Box\ (init\ w);\ \Box\ (init\ w) = \Box\ (init\ w)$
proof −
have *1*: $\vdash (\Box\ (init\ w)) = ((init\ w) \wedge (\ empty\ \vee\ \bigcirc(\Box\ (init\ w))))$
by (*rule BoxEqvAndEmptyOrNextBox*)
hence *2*: $\vdash (\Box\ (init\ w);\ \Box\ (init\ w)) =$
$((init\ w) \wedge ((\ empty\ \vee\ \bigcirc(\Box\ (init\ w)));\ \Box\ (init\ w)))$
by (*metis StateAndChop inteq-reflection*)
have *3*: $\vdash ((\ empty\ \vee\ \bigcirc(\Box\ (init\ w)));\ \Box\ (init\ w)) =$
$(\Box\ (init\ w) \vee\ \bigcirc(\Box\ (init\ w);\ \Box\ (init\ w)))$
by (*rule EmptyOrNextChopEqv*)
have *4*: $\vdash (\Box\ (init\ w);\ \Box\ (init\ w)) =$
$((init\ w) \wedge (\Box\ (init\ w) \vee\ \bigcirc(\Box\ (init\ w);\ \Box\ (init\ w))))$
using *2 3* **by** *fastforce*
have *5*: $\vdash \neg\ (\Box\ (init\ w)) \longrightarrow \neg\ \ (init\ w) \vee\ \neg(\ \bigcirc(\Box\ (init\ w)))$
by (*rule NotBoxImpNotOrNotNextBox*)
have *6*: $\vdash (\Box\ (init\ w);\ \Box\ (init\ w)) \wedge \neg(\ \Box\ (init\ w)) \longrightarrow$
$\bigcirc(\Box\ (init\ w);\ \Box\ (init\ w)) \wedge \neg(\ \bigcirc(\Box\ (init\ w)))$
using *4 5* **by** *fastforce*
hence *7*: $\vdash \Box\ (init\ w);\ \Box\ (init\ w) \wedge finite \longrightarrow \Box\ (init\ w)$
by (*rule NextContra*)
have *8*: $\vdash \Box\ (init\ w);\ \Box\ (init\ w) \wedge inf \longrightarrow \Box\ (init\ w)$
by (*rule BoxStateChopBoxAndInfImpBox*)
have *9*: $\vdash \Box\ (init\ w);\ \Box\ (init\ w) \wedge (finite \vee inf) \longrightarrow \Box\ (init\ w)$
using *7 8* **by** *fastforce*
hence *10*: $\vdash\ \Box\ (init\ w);\ \Box\ (init\ w) \longrightarrow \Box\ (init\ w)$
using *FiniteOrInfinite* **by** *fastforce*
have *11*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge \Box\ (init\ w))$
by (*rule BoxEqvAndBox*)
have *12*: $\vdash empty\ ;\ \Box\ (init\ w) = \Box\ (init\ w)$
by (*rule EmptyChop*)

have *13*: $\vdash ((init\ w) \wedge\ empty\ );\ \Box\ (init\ w) = ((init\ w) \wedge (\ empty\ ;\ \Box\ (init\ w)))$
by (*rule StateAndChop*)
have *14*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge\ empty\ );\ \Box\ (init\ w)$
using *11 12 13* **by** *fastforce*
have *15*: $\vdash (init\ w) \wedge\ empty\ \longrightarrow \Box\ (init\ w)$
by (*rule StateAndEmptyImpBoxState*)
hence *16*: $\vdash ((init\ w) \wedge\ empty\ );\ \Box\ (init\ w) \longrightarrow \Box\ (init\ w);\ \Box\ (init\ w)$
by (*rule LeftChopImpChop*)
have *17*: $\vdash \Box\ (init\ w) \longrightarrow \Box\ (init\ w);\ \Box\ (init\ w)$
using *14 16* **by** *fastforce*
from *10 17* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBoxStateImpBoxYieldsNotBox*:
$\vdash\ \ \neg(\ \Box\ (init\ w)) \longrightarrow (\Box\ (init\ w))\ yields\ (\neg(\ \Box\ (init\ w)))$
proof −
have *1*: $\vdash \Box\ (init\ w);\ \Box\ (init\ w) = \Box\ (init\ w)$ **by** (*rule BoxStateChopBoxEqvBox*)
have *2*: $\vdash \Box\ (init\ w) = (\neg\ \neg(\ \Box\ (init\ w)))$ **by** *auto*
hence *3*: $\vdash \Box\ (init\ w);\ \Box\ (init\ w) = \Box\ (init\ w);\ (\neg\ \neg(\ \Box\ (init\ w)))$ **by** (*rule RightChopEqvChop*)
have *4*: $\vdash \neg(\ \Box\ (init\ w)) \longrightarrow \neg\ (\Box\ (init\ w);\ (\neg\ \neg\ (\Box\ (init\ w))))$ **using** *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *StateEqvBi*:
$\vdash\ \ (init\ w) = bi\ \ (init\ w)$
proof −
have *1*: $\vdash (init\ w) \longrightarrow bi\ \ (init\ w)$ **by** (*rule StateImpBi*)
have *2*: $\vdash bi\ \ (init\ w) \longrightarrow (init\ w)$ **by** (*rule BiElim*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteChopEqvDiamond*:
$\vdash\ \ finite;\ f = \Diamond\ f$
by (*simp add*: *sometimes-d-def*)

3.3.6 Properties of Da and Ba

lemma *DaEqvDtDi*:
$\vdash\ \ \ da\ \ f = \Diamond\ (di\ \ f)$
proof −
have *1*: $\vdash finite;\ (f;\ \#True) = finite;\ (f;\ \#True)$ **by** *auto*
hence *2*: $\vdash finite;\ (f;\ \#True) = finite;\ \ di\ \ f$ **by** (*simp add*: *di-d-def*)
have *3*: $\vdash finite;\ \ di\ \ f = \Diamond(\ di\ \ f)$ **by** (*rule FiniteChopEqvDiamond*)
have *4*: $\vdash finite;\ (f;\ \#True) = \Diamond(\ di\ \ f)$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*da-d-def*)
qed

lemma *DaEqvDiDt*:
$\vdash\ \ \ da\ \ f =\ \ di\ (\Diamond\ f)$
proof −

have *1*: $\vdash$ *finite*; $f = \Diamond\ f$ **by** (*rule FiniteChopEqvDiamond*)
hence *2*: $\vdash$ (*finite*; f); $\#True = (\Diamond\ f)$; $\#True$ **by** (*rule LeftChopEqvChop*)
hence *3*: $\vdash$ (*finite*; f); $\#True = \ di(\ \Diamond\ f)$ **by** (*simp add*: *di-d-def*)
have *4*: $\vdash$ *finite*; (f; $\#True$) = (*finite*; f); $\#True$ **by** (*rule ChopAssoc*)
have *5*: $\vdash$ *finite*; (f; $\#True$) = $\ di\ (\Diamond\ f)$ **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *da-d-def*)
qed

lemma *DtDiEqvDiDt*:
$\vdash\ \ \Diamond\ (di\ \ f) = \ di\ (\Diamond\ f)$
by (*metis ChopAssoc di-d-def sometimes-d-def*)

lemma *DiamondNotEqvNotBox*:
$\vdash \Diamond\ (\neg\ \ f) = (\neg\ (\Box\ f))$
by (*simp add*: *always-d-def*)

lemma *BaEqvBiBt*:
$\vdash\ \ \ ba\ f = bi(\ \Box\ f)$
proof −
have *1*: $\vdash\ \ da\ (\neg\ \ f) = \ di(\ \Diamond\ (\neg\ \ f))$ **by** (*rule DaEqvDiDt*)
have *2*: $\vdash \Diamond\ (\neg\ \ f) = (\neg(\ \Box\ f))$ **by** (*rule DiamondNotEqvNotBox*)
hence *3*: $\vdash\ \ di\ (\Diamond(\neg\ \ f)) = \ di\ (\neg\ (\Box\ f))$ **by** (*rule DiEqvDi*)
have *4*: $\vdash\ \ da\ (\neg\ \ f) = \ di\ (\neg(\ \Box\ f))$ **using** *1 3* **by** *fastforce*
hence *5*: $\vdash\ (\neg\ \ (da\ (\neg\ \ f))) = (\neg\ (\ di\ (\neg(\ \Box\ f))))$ **by** *auto*
hence *6*: $\vdash\ (\neg\ (\ da\ (\neg\ \ f))) = bi(\ \Box\ f)$ **by** (*simp add*: *bi-d-def*)
from *6* **show** *?thesis* **by** (*simp add*: *ba-d-def*)
qed

lemma *DiNotEqvNotBi*:
$\vdash\ \ \ di\ (\neg\ \ f) = (\neg(\ bi\ \ f))$
proof −
have *1*: $\vdash\ bi\ \ f = (\neg\ (\ di\ (\neg\ \ f)))$ **by** (*simp add*: *bi-d-def*)
from *1* **show** *?thesis* **by** *auto*
qed

lemma *NotDiamondNotEqvBox*:
$\vdash\ (\neg\ (\Diamond(\neg\ f))) = \Box\ f$
by (*simp add*: *always-d-def*)

lemma *BaEqvBtBi*:
$\vdash\ \ \ ba\ \ f = \Box\ (bi\ \ f)$
proof −
have *1*: $\vdash\ \ da\ (\neg\ \ f) = \Diamond\ (di\ (\neg\ \ f))$ **by** (*rule DaEqvDtDi*)
have *2*: $\vdash\ \ di\ (\neg\ \ f) = (\neg\ (bi\ \ f))$ **by** (*rule DiNotEqvNotBi*)
hence *3*: $\vdash \Diamond\ (di\ (\neg\ \ f)) = \Diamond(\neg\ (bi\ \ f))$ **by** (*rule DiamondEqvDiamond*)
have *4*: $\vdash\ (\neg\ (\Diamond(\neg\ (bi\ \ f)))) = \Box(bi\ \ f)$ **by** (*rule NotDiamondNotEqvBox*)
have *5*: $\vdash\ (\neg\ (\ da\ (\neg\ \ f))) = \Box(bi\ \ f)$ **using** *1 2 3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *ba-d-def*)
qed

lemma *BtBiEqvBiBt*:
$\vdash \ \Box\ (bi\ \ f) = bi(\ \Box\ f)$
proof −
have *1*: $\vdash \quad ba\ \ f = \Box\ (bi\ \ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash \quad ba\ \ f = bi(\ \Box\ f)$ **by** (*rule BaEqvBiBt*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvBaBoxState*:
$\vdash \ \Box\ (init\ w) = \ \ ba\ (\Box\ (init\ w))$
proof −
have *1*: $\vdash (init\ w) = bi\ \ (init\ w)$ **by** (*rule StateEqvBi*)
hence *2*: $\vdash \Box\ (init\ w) = \Box\ (bi\ \ (init\ w))$ **by** (*rule BoxEqvBox*)
have *3*: $\vdash \Box\ (bi\ \ (init\ w)) = bi(\ \Box\ (init\ w))$ **by** (*rule BtBiEqvBiBt*)
have *4*: $\vdash \Box\ (init\ w) = \Box(\Box\ (init\ w))$ **by** (*rule BoxEqvBoxBox*)
hence *5*: $\vdash bi(\ \Box\ (init\ w)) = bi\ (\Box(\Box\ (init\ w)))$ **by** (*rule BiEqvBi*)
have *6*: $\vdash \ \ ba(\ \Box\ (init\ w)) = bi(\ \Box(\Box\ (init\ w)))$ **by** (*rule BaEqvBiBt*)
from *2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBi*:
$\vdash \quad ba\ \ f \longrightarrow bi\ \ f$
proof −
have *1*: $\vdash \ \ ba\ \ f = \Box(bi\ \ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash \Box(bi\ \ f) \longrightarrow bi\ \ f$ **by** (*rule BoxElim*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *BaImpBt*:
$\vdash \quad ba\ \ f \longrightarrow \Box\ f$
proof −
have *1*: $\vdash \ \ ba\ \ f = bi(\ \Box\ f)$ **by** (*rule BaEqvBiBt*)
have *2*: $\vdash bi(\ \Box\ f) \longrightarrow \Box\ f$ **by** (*rule BiElim*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *DiamondImpDa*:
$\vdash \ \ \Diamond\ f \longrightarrow \ \ da\ \ f$
by (*metis DiIntro DiamondImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *DiImpDa*:
$\vdash \quad di\ \ f \longrightarrow \ \ da\ \ f$
by (*metis NowImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *BoxAndChopImport*:
$\vdash \ \ \Box\ h \wedge f;\ g \longrightarrow f;\ (h \wedge g)$
proof −
have *1*: $\vdash h \longrightarrow g \longrightarrow (h\wedge\ g)$ **by** *auto*
hence *2*: $\vdash \Box\ h \longrightarrow \Box(g \longrightarrow (h\wedge\ g))$ **by** (*rule ImpBoxRule*)
have *3*: $\vdash \Box(g \longrightarrow (h\wedge\ g)) \longrightarrow f;\ g \longrightarrow f;\ (h\wedge\ g)$ **by** (*rule BoxChopImpChop*)

from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndChopImport*:
$\vdash$ *ba* $f \wedge (g;\ g1) \longrightarrow (f \wedge g);\ (f \wedge g1)$
proof −
have *1*: $\vdash$ *ba* $f \longrightarrow$ *bi* f **by** (*rule BaImpBi*)
have *2*: $\vdash$ *bi* $f \wedge (g;\ g1) \longrightarrow (f \wedge g);\ g1$ **by** (*rule BiAndChopImport*)
have *3*: $\vdash$ *ba* $f \longrightarrow \Box\ f$ **by** (*rule BaImpBt*)
have *4*: $\vdash \Box\ f \wedge (f \wedge g);\ g1 \longrightarrow (f \wedge g);\ (f \wedge g1)$ **by** (*rule BoxAndChopImport*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopAndCommute*:
$\vdash$ $f;\ (g \wedge g1) = f;\ (g1 \wedge g)$
proof −
have *1*: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightChopEqvChop*)
qed

lemma *ChopAndA*:
$\vdash$ $f;\ (g \wedge g1) \longrightarrow f;\ g$
proof −
have *1*: $\vdash (g \wedge g1) \longrightarrow g$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightChopImpChop*)
qed

lemma *ChopAndB*:
$\vdash$ $f;\ (g \wedge g1) \longrightarrow f;\ g1$
proof −
have *1*: $\vdash (g \wedge g1) \longrightarrow g1$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightChopImpChop*)
qed

lemma *BoxStateAndChopEqvChop*:
$\vdash$ $(\Box\ (init\ w) \wedge (f;\ g)) = ((\Box\ (init\ w) \wedge f);\ (\Box\ (init\ w) \wedge g))$
proof −
have *1*: $\vdash \Box\ (init\ w) =$ *ba*$(\ \Box\ (init\ w))$
by (*rule BoxStateEqvBaBoxState*)
have *2*: $\vdash$ *ba*$(\ \Box\ (init\ w)) \wedge (f;\ g) \longrightarrow (\Box\ (init\ w) \wedge f);\ (\Box\ (init\ w) \wedge g)$
by (*rule BaAndChopImport*)
have *3*: $\vdash \Box\ (init\ w) \wedge (f;\ g) \longrightarrow (\Box\ (init\ w) \wedge f);\ (\Box\ (init\ w) \wedge g)$
using *1 2* **by** *fastforce*
have *11*: $\vdash (\Box\ (init\ w) \wedge f);\ (\Box\ (init\ w) \wedge g) \longrightarrow (\Box\ (init\ w));\ (\Box\ (init\ w) \wedge g)$
by (*rule AndChopA*)
have *12*: $\vdash (\Box\ (init\ w));\ (\Box\ (init\ w) \wedge g) \longrightarrow (\Box\ (init\ w));\ (\Box\ (init\ w))$
by (*rule ChopAndA*)
have *13*: $\vdash (\Box\ (init\ w));\ (\Box\ (init\ w)) = \Box\ (init\ w)$
by (*rule BoxStateChopBoxEqvBox*)
have *14*: $\vdash (\Box\ (init\ w) \wedge f);\ (\Box\ (init\ w) \wedge g) \longrightarrow f;\ (\Box\ (init\ w) \wedge g)$

by (*rule AndChopB*)
have *15*: $\vdash f;\ (\Box\ (init\ w) \wedge g) \longrightarrow f;\ g$
by (*rule ChopAndB*)
have *16*: $\vdash (\Box\ (init\ w) \wedge f);\ (\Box\ (init\ w) \wedge g) \longrightarrow \Box\ (init\ w) \wedge (f;\ g)$
using *11 12 13 14 15* **by** *fastforce*
from *3 16* **show** *?thesis* **by** *fastforce*
qed

lemma *DiEqvNotBiNot*:
$\vdash\ \ di\ \ f = (\neg(\ bi\ (\neg\ \ f)))$
proof −
have *1*: $\vdash bi\ (\neg\ \ f) = (\neg\ (\ di\ (\neg\ \neg\ \ f)))$ **by** (*simp add*: *bi-d-def*)
hence *2*: $\vdash\ \ di\ (\neg\ \neg\ \ f) = (\neg(\ bi\ (\neg\ \ f)))$ **by** *auto*
have *3*: $\vdash f = (\neg\ \neg\ \ f)$ **by** *auto*
hence *4*: $\vdash\ \ di\ \ f =\ \ di\ \ (\neg\ \neg\ \ f)$ **by** (*rule DiEqvDi*)
from *2 4* **show** *?thesis* **by** *auto*
qed

lemma *ChopAndBoxImport*:
$\vdash\ \ f;\ g \wedge \Box\ h \longrightarrow f;\ (g \wedge h)$
proof −
have *1*: $\vdash \Box\ h \wedge f;\ g \longrightarrow f;\ (h \wedge g)$ **by** (*rule BoxAndChopImport*)
have *2*: $\vdash f;\ (h \wedge g) = f;\ (g \wedge h)$ **by** (*rule ChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *AndChopAndCommute*:
$\vdash\ \ (f \wedge g);\ (f1 \wedge g1) = (g \wedge f);\ (g1 \wedge f1)$
proof −
have *1*: $\vdash (f \wedge g);\ (f1 \wedge g1) = (g \wedge f);\ (f1 \wedge g1)$ **by** (*rule AndChopCommute*)
have *2*: $\vdash (g \wedge f);\ (f1 \wedge g1) = (g \wedge f);\ (g1 \wedge f1)$ **by** (*rule ChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopImpChop*:
assumes $\vdash f \longrightarrow f1$
$\vdash g \longrightarrow g1$
shows $\vdash f;g \longrightarrow f1;g1$
proof −
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ g \longrightarrow f1;\ g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence *4*: $\vdash f1;\ g \longrightarrow f1;\ g1$ **by** (*rule RightChopImpChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvChop*:
assumes $\vdash f = f1$
$\vdash g = g1$
shows $\vdash f;g = f1;g1$

proof −
have *1*: ⊢ $f = f1$ **using** *assms* **by** *auto*
hence *2*: ⊢ $f;\ g = f1;\ g$ **by** (*rule LeftChopEqvChop*)
have *3*: ⊢ $g = g1$ **using** *assms* **by** *auto*
hence *4*: ⊢ $f1;\ g = f1;\ g1$ **by** (*rule RightChopEqvChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxImpBox*:
⊢ $\Box\ h \longrightarrow \Box(g \longrightarrow \Box\ h \wedge g\ )$
proof −
have *1*: ⊢ $\Box\ h \longrightarrow (g \longrightarrow \Box\ h \wedge g\ )$ **by** *auto*
hence *2*: ⊢ $\Box(\Box\ h) \longrightarrow \Box(g \longrightarrow \Box\ h \wedge g\ )$ **by** (*rule ImpBoxRule*)
have *3*: ⊢ $\Box\ h = \Box(\Box h)$ **by** (*rule BoxEqvBoxBox*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxChopImpChopBox*:
⊢ $\Box\ h \longrightarrow f;\ g \longrightarrow f;\ (\Box\ h \wedge g)$
proof −
have *1*: ⊢ $\Box\ h \longrightarrow \Box(g \longrightarrow \Box\ h \wedge g\ )$ **by** (*rule BoxImpBoxImpBox*)
have *2*: ⊢ $\Box(g \longrightarrow \Box\ h \wedge g\ ) \longrightarrow f;\ g \longrightarrow f;\ (\Box\ h \wedge g)$ **by** (*rule BoxChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotChopEqvYieldsNot*:
⊢ $(\neg\ (f;\ g)) = f\ yields\ (\neg\ \ g)$
proof −
have *1*: ⊢ $g = (\neg\ \neg\ \ g)$ **by** *auto*
hence *2*: ⊢ $f;\ g = f;\ (\neg\ \neg\ \ g)$ **by** (*rule RightChopEqvChop*)
hence *3*: ⊢ $(\neg\ (f;\ g)) = (\neg\ (f;\ (\neg\ \neg\ \ g)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *NotDiFalse*:
⊢ $\neg\ (\ di\ \ \#False)$
proof −
have *1*: ⊢ $(init\ \#True) \longrightarrow bi\ (init\ \#True)$ **by** (*rule StateImpBi*)
hence *2*: ⊢ $\#True \longrightarrow bi\ \#True$ **by** (*simp add*: *BiGen*)
have *3*: ⊢ $\#True$ **by** *auto*
have *4*: ⊢ $bi\ \#True$ **using** *2 3 MP* **by** *auto*
hence *5*: ⊢ $\neg\ (\ di\ (\neg\ \#True))$ **by** (*simp add*: *bi-d-def*)
have *6*: ⊢ $(\neg\ \#True) =\ \#False$ **by** *auto*
hence *7*: ⊢ $\ di\ (\neg\ \#True) =\ \ di\ \ \#False$ **by** (*rule DiEqvDi*)
from *5 7* **show** *?thesis* **by** *auto*
qed

lemma *StateAndEmptyChop*:
⊢ $((init\ w) \wedge\ \ empty\ );\ f = ((init\ w) \wedge f)$
proof −

have *1*: $\vdash ((init\ w) \wedge \ empty\ ); f = ((init\ w) \wedge \ empty\ ; f)$ **by** (*rule StateAndChop*)
have *2*: $\vdash \ empty\ ; f = f$ **by** (*rule EmptyChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndNextChop*:
$\vdash \ ((init\ w) \wedge \bigcirc f); g = ((init\ w) \wedge \bigcirc(f; g))$
proof −
have *1*: $\vdash ((init\ w) \wedge \bigcirc f); g = ((init\ w) \wedge (\bigcirc f); g)$ **by** (*rule StateAndChop*)
have *2*: $\vdash (\bigcirc f); g = \bigcirc(f; g)$ **by** (*rule NextChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NextAndEqvNextAndNext*:
$\vdash \bigcirc (f \wedge g) = (\bigcirc f \wedge \bigcirc g)$
by (*metis NextAndNextEqvNextRule int-eq lift-and-com*)

lemma *NextStateAndChop*:
$\vdash \ \bigcirc(((init\ w) \wedge f); g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f; g))$
proof −
have *1*: $\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge f; g)$ **by** (*rule StateAndChop*)
hence *2*: $\vdash \bigcirc(((init\ w) \wedge f); g) = \bigcirc((init\ w) \wedge f; g)$ **by** (*rule NextEqvNext*)
have *3*: $\vdash \bigcirc((init\ w) \wedge f; g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f; g))$ **by** (*rule NextAndEqvNextAndNext*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateYieldsEqv*:
$\vdash \ ((init\ w) \longrightarrow (f\ yields\ \ g)) = ((init\ w) \wedge f)\ yields\ \ g$
proof −
have *1*: $\vdash ((init\ w) \wedge f); (\neg\ \ g) = ((init\ w) \wedge f; (\neg\ \ g))$ **by** (*rule StateAndChop*)
hence *2*: $\vdash ((init\ w) \longrightarrow \neg\ (f; (\neg\ \ g))) = (\neg\ (((init\ w) \wedge f); (\neg\ \ g)\ ))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *StateAndDi*:
$\vdash \ ((init\ w) \wedge \ di\ \ f) = \ di\ ((init\ w) \wedge f)$
proof −
have *1*: $\vdash ((init\ w) \wedge f); \#True = \ ((init\ w) \wedge f; \#True)$ **by** (*rule StateAndChop*)
from *1* **show** *?thesis* **by** (*metis di-d-def inteq-reflection*)
qed

lemma *DiNext*:
$\vdash \ \ di(\ \bigcirc f) = \bigcirc\ (di\ \ f)$
proof −
have *1*: $\vdash (\bigcirc f); \#True = \bigcirc(f; \#True)$ **by** (*rule NextChop*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiNextState*:
$\vdash \ \ di(\ \bigcirc\ (init\ w)) = \bigcirc\ (init\ w)$

proof −
have *1*: ⊢ *di*(○ (*init w*)) = ○(*di* (*init w*)) **by** (*rule DiNext*)
have *2*: ⊢ *di* (*init w*) = (*init w*) **by** (*rule DiState*)
hence *3*: ⊢ ○(*di* (*init w*)) = ○ (*init w*) **by** (*rule NextEqvNext*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpBiGen*:
assumes ⊢ (*init w*) ⟶ *f*
shows ⊢ (*init w*) ⟶ *bi f*
proof −
have *1*: ⊢ (*init w*) ⟶ *f* **using** *assms* **by** *auto*
hence *2*: ⊢ ¬ *f* ⟶ ¬ (*init w*) **by** *auto*
hence *3*: ⊢ *di* (¬ *f*) ⟶ *di* (¬ (*init w*)) **by** (*rule DiImpDi*)
hence *4*: ⊢ *di* (¬ *f*) ⟶ *di* (*init* (¬*w*)) **by** (*metis Initprop*(*2*) *inteq-reflection*)
have *5*: ⊢ *di* (*init* (¬ *w*)) = (*init* (¬ *w*)) **by** (*rule DiState*)
have *6*: ⊢ *di* (¬ *f*) ⟶ ¬ (*init w*) **using** *4 5* **using** *Initprop*(*2*) **by** *fastforce*
hence *7*: ⊢ (*init w*) ⟶ ¬ (*di* (¬ *f*)) **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *ChopAndNotChopImp*:
⊢ *f*; *g* ∧ ¬ (*f*; *g1*) ⟶ *f*; (*g* ∧ ¬ *g1*)
proof −
have *1*: ⊢ *g* ⟶ (*g*∧ ¬ *g1*) ∨ *g1* **by** *auto*
hence *2*: ⊢ *f*; *g* ⟶ *f*; ((*g*∧ ¬ *g1*) ∨ *g1*) **by** (*rule RightChopImpChop*)
have *3*: ⊢ *f*; ((*g*∧ ¬ *g1*) ∨ *g1*) ⟶ (*f*; (*g*∧ ¬ *g1*)) ∨ (*f*; *g1*) **by** (*rule ChopOrImp*)
have *4*: ⊢ *f*; *g* ⟶ *f*; (*g*∧ ¬ *g1*) ∨ *f*; *g1* **using** *2 3 MP* **by** *fastforce*
from *4* **show** *?thesis* **by** *auto*
qed

lemma *ChopAndYieldsImp*:
⊢ *f*; *g* ∧ *f yields* *g1* ⟶ *f*; (*g* ∧ *g1*)
proof −
have *1*: ⊢ *g* ⟶ (*g*∧ *g1*) ∨ ¬ *g1* **by** *auto*
hence *2*: ⊢ *f*; *g* ⟶ *f*; ((*g*∧ *g1*) ∨ ¬ *g1*) **by** (*rule RightChopImpChop*)
have *3*: ⊢ *f*; ((*g*∧ *g1*) ∨ ¬ *g1*) ⟶ (*f*; (*g*∧ *g1*)) ∨ (*f*; (¬ *g1*)) **by** (*rule ChopOrImp*)
have *4*: ⊢ *f*; *g* ⟶ *f*; (*g*∧ *g1*) ∨ *f*; (¬ *g1*) **using** *2 3 MP* **by** *fastforce*
hence *5*: ⊢ *f*; *g* ∧ ¬ (*f*; (¬ *g1*)) ⟶ *f*; (*g* ∧ *g1*) **by** *auto*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ChopAndYieldsMP*:
⊢ *f*; *g* ∧ *f yields* (*g*⟶ *g1*) ⟶ *f*; *g1*
proof −
have *1*: ⊢ *f*; *g* ∧ *f yields* (*g*⟶ *g1*) ⟶ *f*; (*g* ∧ (*g* ⟶ *g1*)) **by** (*rule ChopAndYieldsImp*)
have *2*: ⊢ *g* ∧ (*g* ⟶ *g1*) ⟶ *g1* **by** *auto*
hence *3*: ⊢ *f*; (*g* ∧ (*g* ⟶ *g1*)) ⟶ *f*; *g1* **by** (*rule RightChopImpChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *OrYieldsImp*:
$\vdash \ (f \vee \ f1) \ yields \ \ g = ((f \ yields \ \ g) \wedge (f1 \ yields \ \ g))$
proof −
have *1*: $\vdash ((f \vee \ f1); (\neg \ \ g)) = ((f; (\neg \ \ g)) \vee \ \ (f1; (\neg \ \ g)))$ **by** (*rule OrChopEqv*)
hence *2*: $\vdash (\neg \ ((f \vee \ f1); (\neg \ \ g))) = (\neg \ (f; (\neg \ \ g)) \wedge \ \neg (f1; (\neg \ \ g)))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *LeftYieldsImpYields*:
assumes $\vdash f \longrightarrow f1$
shows $\vdash (f1 \ yields \ \ g) \longrightarrow (f \ yields \ \ g)$
proof −
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f; (\neg \ \ g) \longrightarrow f1; (\neg \ \ g)$ **by** (*rule LeftChopImpChop*)
hence *3*: $\vdash \neg \ (f1; (\neg \ \ g)) \longrightarrow \neg \ (f; (\neg \ \ g))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *LeftYieldsEqvYields*:
assumes $\vdash \ f = f1$
shows $\vdash (f \ yields \ \ g) = (f1 \ yields \ \ g)$
proof −
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f; (\neg \ \ g) = f1; (\neg \ \ g)$ **by** (*rule LeftChopEqvChop*)
hence *3*: $\vdash (\neg \ (f; (\neg \ \ g))) = (\neg \ (f1; (\neg \ \ g)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

3.3.7 Properties of Fin

lemma *FinEqvTrueChopAndEmpty*:
$\vdash \ fin \ f = \#True;(f \wedge empty)$
proof −
have *1*: $\vdash fin \ f = \Box(empty \longrightarrow f)$
by (*simp add*: *fin-d-def*)
have *2*: $\vdash \Box(empty \longrightarrow f) = (\neg(\Diamond(\neg(empty \longrightarrow f \ ) \ ) \ ))$
by (*simp add*: *always-d-def*)
have *3*: $\vdash (\neg(empty \longrightarrow f \ )) = (\neg \ f \wedge empty)$
by *auto*
hence *4*: $\vdash \Diamond(\neg(empty \longrightarrow f \ )) = \Diamond(\neg \ f \wedge empty)$
using *DiamondEqvDiamond* **by** *blast*
hence *5*: $\vdash \neg(\Diamond(\neg(empty \longrightarrow f \ ))) = (\neg(\Diamond(\neg \ f \wedge empty)))$
by *auto*
have *51*: $\vdash finite;((\neg \ f \wedge empty) \wedge finite) \ \longrightarrow finite \ ; \ \ (\neg \ f \wedge empty)$
by (*simp add*: *ChopAndA*)
have *52*: $\vdash (\#True;(\neg \ f \wedge empty) \wedge finite) \longrightarrow finite \ ; \ \ (\neg \ f \wedge empty)$
by (*metis 51 TrueChopAndFiniteEqvAndFiniteChopFinite int-eq*)
have *53*: $\vdash \neg \ (\#True;(f \wedge empty)) \ \longrightarrow finite \ ; \ \ (\neg \ f \wedge empty)$
by (*metis 52 Finprop*(*5*) *int-eq*)

have *54*: $\vdash \neg$ *finite*;$(\neg\ f \wedge$ *empty*$) \longrightarrow \#$*True*;$(f \wedge$ *empty*$)$
using *53* **by** *auto*
have *6*: $\vdash (\neg(\Diamond(\neg\ f \wedge$ *empty*$))) \longrightarrow \#$*True*;$(f \wedge$ *empty*$)$
unfolding *sometimes-d-def* **using** *54* **by** *auto*
have *61*: $\vdash \neg\ f \wedge$ *empty* $\longrightarrow$ *finite*
by (*metis ChopAndB DiamondEmptyEqvFinite NowImpDiamond inteq-reflection lift-imp-trans sometimes-d-def*)
have *62*: $\vdash(\neg\ \#$*True*;$(f \wedge$ *empty*$)) =$ *finite*;$(\neg\ f \wedge$ *empty*$)$
using *61*
by (*metis* (*no-types*) *Finprop*(*5*) *Prop10 TrueChopAndFiniteEqvAndFiniteChopFinite inteq-reflection*)
have *7*: $\vdash \#$*True*;$(f \wedge$ *empty*$) \longrightarrow (\neg(\Diamond(\neg\ f \wedge$ *empty*$)))$
unfolding *sometimes-d-def* **using** *TrueChopAndFiniteEqvAndFiniteChopFinite*[*of LIFT*$(f \wedge$ *empty*$)$]
using *62* **by** *auto*
have *8*: $\vdash (\neg(\Diamond(\neg\ f \wedge$ *empty*$))) = \#$*True*;$(f \wedge$ *empty*$)$
by (*simp add*: *6 7 int-iffI*)
from *1 2 5 8* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondFin*:
$\vdash \Diamond($*fin w*$) =$ *fin w*
by (*metis* (*no-types*, *lifting*) *ChopAssoc ChopOrEqv FinEqvTrueChopAndEmpty FiniteChopFiniteEqvFinite FiniteChopInfEqvInf FiniteOrInfinite int-eq-true inteq-reflection sometimes-d-def*)

lemma *FiniteChopFinExportA*:
$\vdash (f \wedge$ *finite*$);(g \wedge$ *fin w*$) \longrightarrow$ *fin w*
using *DiamondFin*
by (*metis ChopAndB FiniteChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *FinImpBox*:
$\vdash$ *fin w* $\longrightarrow \Box($*fin w*$)$
by (*metis BoxImpBoxBox fin-d-def*)

lemma *FinAndChopImport*:
$\vdash ($*fin w*$) \wedge (f;g) \longrightarrow f;(($*fin w*$) \wedge g)$
proof −
have *1*: $\vdash$ *fin w* $\longrightarrow \Box($*fin w*$)$ **by** (*rule FinImpBox*)
hence *2*: $\vdash$ *fin w* $\wedge f;g \longrightarrow \Box($*fin w*$) \wedge (f;g)$ **by** *auto*
have *3*: $\vdash \Box($*fin w*$) \wedge (f;g) \longrightarrow f;(($*fin w*$) \wedge g)$ **using** *BoxAndChopImport* **by** *blast*
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *FinAndChop*:
$\vdash ((f \wedge$ *finite*$);(g \wedge$ *fin w*$)) = ($*fin w*$ \wedge (f \wedge$ *finite*$);g)$
using *FinAndChopImport FiniteChopFinExportA ChopAndA ChopAndCommute*
by *fastforce*

lemma *ChopAndEmptyEqvEmptyChopEmpty*:
$\vdash ((f;g) \wedge$ *empty*$) = (f \wedge$ *empty*$);(g \wedge$ *empty*$)$
by (*auto simp*: *itl-defs min-absorb1*)

lemma *FinAndEmpty*:
$\vdash ((\mathit{fin}\ w) \wedge \mathit{empty}) = (w \wedge \mathit{empty})$
proof −
have *1*: $\vdash ((\mathit{fin}\ w) \wedge \mathit{empty}) = (\#\mathit{True};(w \wedge \mathit{empty}) \wedge \mathit{empty})$
using *FinEqvTrueChopAndEmpty* **by** *fastforce*
have *2*: $\vdash (\#\mathit{True};(w \wedge \mathit{empty}) \wedge \mathit{empty}) = ((\#\mathit{True} \wedge \mathit{empty});(w \wedge \mathit{empty}))$
using *ChopAndEmptyEqvEmptyChopEmpty*[*of* $\mathit{LIFT}(\#\mathit{True})$ $\mathit{LIFT}(w \wedge \mathit{empty})$]
by (*metis AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty Prop11 Prop12 int-eq*)
have *3*: $\vdash (\#\mathit{True} \wedge \mathit{empty});(w \wedge \mathit{empty}) = (\mathit{empty};(w \wedge \mathit{empty}))$
using *LeftChopEqvChop* **by** *fastforce*
have *4*: $\vdash (\mathit{empty};(w \wedge \mathit{empty})) = (w \wedge \mathit{empty})$
using *EmptyChop* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *AndFinEqvChopAndEmpty*:
$\vdash \ ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ g) = (f \wedge \mathit{finite});\ (g \wedge \ \mathit{empty}\ )$
proof −
have *1*: $\vdash ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ g) = ((f \wedge \mathit{finite})\ ;\mathit{empty} \wedge \mathit{fin}\ g)$
using *ChopEmpty* **by** (*metis inteq-reflection*)
have *2*: $\vdash (\mathit{fin}\ g \wedge (f \wedge \mathit{finite});\mathit{empty}) = ((f \wedge \mathit{finite});(\mathit{empty} \wedge \mathit{fin}\ g))$
using *FinAndChop* **by** *fastforce*
have *3*: $\vdash (\mathit{empty} \wedge \mathit{fin}\ g) = \ (\mathit{fin}\ g \wedge \mathit{empty})$
by *auto*
have *4*: $\vdash (\mathit{fin}\ g \wedge \mathit{empty}) = (g \wedge \mathit{empty})$
using *FinAndEmpty* **by** *metis*
have *5*: $\vdash (\mathit{empty} \wedge \mathit{fin}\ g) = (g \wedge \mathit{empty})$
using *3 4* **by** *auto*
hence *6*: $\vdash (f \wedge \mathit{finite});(\mathit{empty} \wedge \mathit{fin}\ g) = (f \wedge \mathit{finite});(g \wedge \mathit{empty})$
using *RightChopEqvChop* **by** *blast*
from *1 2 5* **show** *?thesis* **by** (*metis inteq-reflection lift-and-com*)
qed

lemma *AndFinEqvChopStateAndEmpty*:
$\vdash \ ((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w)) = (f \wedge \mathit{finite});\ ((\mathit{init}\ w) \wedge \ \mathit{empty}\ )$
using *AndFinEqvChopAndEmpty* **by** *blast*

lemma *FinStateEqvStateAndEmptyOrNextFinState*:
$\vdash \ \mathit{fin}\ \ (\mathit{init}\ w) = (((\mathit{init}\ w) \wedge \ \mathit{empty}\ ) \vee \ \bigcirc(\ \mathit{fin}\ \ (\mathit{init}\ w)))$
proof −
have *1*: $\vdash \mathit{fin}\ (\mathit{init}\ w) = \Box(\ \mathit{empty} \longrightarrow \mathit{init}\ w)$
by (*simp add*: *fin-d-def*)
have *2* : $\vdash \Box(\mathit{empty} \longrightarrow \mathit{init}\ w) =$
$((\mathit{empty} \longrightarrow \mathit{init}\ w) \wedge \mathit{wnext}\ (\Box\ (\mathit{empty} \longrightarrow \mathit{init}\ w)))$
by (*rule BoxEqvAndWnextBox*)
have *3*: $\vdash \mathit{fin}\ (\mathit{init}\ w) = \ ((\mathit{empty} \longrightarrow \mathit{init}\ w) \wedge \mathit{wnext}\ (\mathit{fin}\ (\mathit{init}\ w)))$
using *1 2* **by** (*simp add*: *fin-d-def*)
have *4*: $\vdash \mathit{wnext}\ (\mathit{fin}\ (\mathit{init}\ w)) = (\mathit{empty} \vee \bigcirc\ (\mathit{fin}\ (\mathit{init}\ w)))$
by (*rule WnextEqvEmptyOrNext*)

have *5*: $\vdash$ *fin* (*init w*) = ((*empty* $\longrightarrow$ *init w*) $\wedge$ (*empty* $\vee$ $\bigcirc$ (*fin* (*init w*))))
using *3 4* **by** *fastforce*
have *6*: $\vdash$ ((*empty* $\longrightarrow$ *init w*) $\wedge$ (*empty* $\vee$ $\bigcirc$ (*fin* (*init w*)))) =
(((*empty* $\longrightarrow$ *init w*) $\wedge$ *empty*) $\vee$ ((*empty* $\longrightarrow$ *init w*) $\wedge$ $\bigcirc$ (*fin* (*init w*))))
by *auto*
have *7*: $\vdash$ ((*empty* $\longrightarrow$ *init w*) $\wedge$ *empty*) = ((*init w*) $\wedge$ *empty*)
by *auto*
have *8*: $\vdash$ ((*empty* $\longrightarrow$ *init w*) $\wedge$ $\bigcirc$ (*fin* (*init w*))) = $\bigcirc$ (*fin* (*init w*))
by (*metis* (*no-types*, *lifting*) *5 DiamondFin NextDiamondImpDiamond Prop10 Prop12 int-eq lift-and-com*)
have *9*: $\vdash$ (((*empty* $\longrightarrow$ *init w*) $\wedge$ *empty*) $\vee$ ((*empty* $\longrightarrow$ *init w*) $\wedge$ $\bigcirc$ (*fin* (*init w*)))) =
((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*fin* (*init w*))
using *7 8* **by** *auto*
from *5 6 8 9* **show** *?thesis* **by** *fastforce*
qed

lemma *FinChopEqvOr*:
$\vdash$ (*fin* (*init w*)); *f* = (((*init w*) $\wedge$ *f*) $\vee$ $\bigcirc$((*fin* (*init w*)); *f*))
proof −
have *1*: $\vdash$ *fin* (*init w*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*fin* (*init w*)))
by (*rule FinStateEqvStateAndEmptyOrNextFinState*)
hence *2*: $\vdash$ (*fin* (*init w*)); *f* = (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$(*fin* (*init w*))); *f*
by (*rule LeftChopEqvChop*)
have *3*: $\vdash$ (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$ (*fin* (*init w*))); *f*
= (((*init w*) $\wedge$ *empty*); *f* $\vee$ ($\bigcirc$ (*fin* (*init w*))); *f*)
by (*rule OrChopEqv*)
have *4*: $\vdash$ ((*init w*) $\wedge$ *empty*); *f* = ((*init w*) $\wedge$ *f*)
by (*rule StateAndEmptyChop*)
have *5*: $\vdash$ ($\bigcirc$ (*fin* (*init w*))); *f* = $\bigcirc$((*fin* (*init w*)); *f*)
by (*rule NextChop*)
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FinChopEqvDiamond*:
$\vdash$ (*fin* (*init w*) $\wedge$ *finite*); *f* = $\Diamond$ ((*init w*) $\wedge$ *f*)
proof −
have *1*: $\vdash$ (*fin* (*init w*) $\wedge$ *finite*) = (*finite*;((*init w*) $\wedge$ *empty*))
by (*metis AndFinEqvChopAndEmpty int-simps*(*17*) *inteq-reflection lift-and-com*)
hence *2*: $\vdash$ (*fin* (*init w*) $\wedge$ *finite*);*f* = (*finite*;((*init w*) $\wedge$ *empty*));*f*
by (*rule LeftChopEqvChop*)
have *3*: $\vdash$ *finite*;(((*init w*) $\wedge$ *empty*);*f*) = (*finite*;((*init w*) $\wedge$ *empty*));*f*
by (*rule ChopAssoc*)
have *4*: $\vdash$ *finite*;(((*init w*) $\wedge$ *empty*);*f*)= $\Diamond$ (((*init w*) $\wedge$ *empty*);*f*)
by (*simp add*: *sometimes-d-def*)
have *5*: $\vdash$ ((*init w*) $\wedge$ *empty*);*f* = ((*init w*) $\wedge$ *f*)
using *StateAndEmptyChop* **by** *blast*
hence *6*: $\vdash$ $\Diamond$ (((*init w*) $\wedge$ *empty*);*f*) = $\Diamond$ ((*init w*) $\wedge$ *f*)
by (*rule DiamondEqvDiamond*)
from *2 3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *NotDiamondAndNot*:
$\vdash \neg(\ \Diamond\ (\ f \wedge \neg\ f))$
proof −
have *1*: $\vdash (\neg(\ \Diamond\ (\ f \wedge \neg\ f))) = \Box(\neg(f \wedge \neg f))$ **using** *NotDiamondNotEqvBox* **by** *fastforce*
have *2*: $\vdash \neg(f \wedge \neg f)$ **by** *simp*
have *3*: $\vdash \Box(\neg(f \wedge \neg f))$ **using** *2* **by** (*simp add*: *BoxGen*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FinYields*:
$\vdash \quad (\ fin\ \ (init\ w) \wedge finite)\ \ yields\ \ (init\ w)$
proof −
have *1*: $\vdash (fin\ (init\ w) \wedge finite);\ (\neg(init\ w)) = \Diamond((init\ w) \wedge \neg(init\ w))$
by (*rule FinChopEqvDiamond*)
have *2*: $\vdash\ \neg(\ \Diamond((init\ w) \wedge \neg\ \ (init\ w)))$
by (*rule NotDiamondAndNot*)
have *3*: $\vdash \neg\ ((\ fin\ \ (init\ w) \wedge finite);\ (\neg\ \ (init\ w)))$
using *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ImpAndFinStateOrFinNotState*:
$\vdash f \longrightarrow (f \wedge fin\ (init\ w)) \vee (f \wedge fin\ (\neg\ (init\ w)))$
by (*simp add*: *itl-defs Valid-def*)

lemma *AndFinChopEqvStateAndChop*:
$\vdash \quad ((f \wedge finite) \wedge fin\ \ (init\ w));\ g = (f \wedge finite);\ ((init\ w) \wedge g)$
proof −
have *1*: $\vdash (\ fin\ \ (init\ w) \wedge finite)\ yields\ \ (init\ w)$
by (*rule FinYields*)
have *2*: $\vdash (f \wedge finite) \wedge\ \ fin\ \ (init\ w) \longrightarrow\ \ fin\ \ (init\ w)$
by *auto*
hence *3*: $\vdash (\ fin\ \ (init\ w) \wedge finite)\ yields\ \ (init\ w) \longrightarrow$
$((f \wedge finite) \wedge\ \ fin\ \ (init\ w))\ yields\ \ (init\ w)$
using *LeftYieldsImpYields*
by (*metis AndFinEqvChopAndEmpty Prop11 Prop12 inteq-reflection*)
have *4*: $\vdash ((f \wedge finite) \wedge\ \ fin\ \ (init\ w))\ yields\ \ (init\ w)$
using *1 3 MP* **by** *fastforce*
have *5*: $\vdash ((f \wedge finite) \wedge\ \ fin\ \ (init\ w));\ g \wedge ((f \wedge finite) \wedge\ \ fin\ \ (init\ w))\ yields\ (init\ w)$
$\longrightarrow ((f \wedge finite) \wedge\ \ fin\ \ (init\ w));\ (g \wedge (init\ w))$
by (*rule ChopAndYieldsImp*)
have *6*: $\vdash ((f \wedge finite) \wedge\ \ fin\ \ (init\ w));\ g \longrightarrow$
$((f \wedge finite) \wedge\ \ fin\ \ (init\ w));\ (g \wedge (init\ w))$
using *4 5* **by** *fastforce*
have *7*: $\vdash ((f \wedge finite) \wedge\ \ fin\ \ (init\ w));\ (g \wedge (init\ w)) \longrightarrow (f \wedge finite);\ (g \wedge (init\ w))$
by (*rule AndChopA*)
have *8*: $\vdash g \wedge (init\ w) \longrightarrow (init\ w) \wedge g$
by *auto*
hence *9*: $\vdash (f \wedge finite);\ (g \wedge (init\ w)) \longrightarrow (f \wedge finite);\ ((init\ w) \wedge g)$

by (*rule RightChopImpChop*)
have *10*: $\vdash ((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w));\ g \longrightarrow (f \wedge \mathit{finite});\ ((\mathit{init}\ w) \wedge g)$
using *6 7 9* **by** *fastforce*
have *11*: $\vdash (f \wedge \mathit{finite}) \longrightarrow ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ w)) \vee ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))$
using *ImpAndFinStateOrFinNotState* **by** *blast*
hence *12*: $\vdash (f \wedge \mathit{finite});\ ((\mathit{init}\ w) \wedge g) \longrightarrow$
$(((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w)) \vee$
$((\mathit{finite} \wedge f) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))\ ));\ ((\mathit{init}\ w) \wedge g)$
using *LeftChopImpChop*
by (*metis inteq-reflection lift-and-com*)
have *13*: $\vdash (((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w)) \vee ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))));\ ((\mathit{init}\ w) \wedge g)$
$=$
$(((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w));$
$((\mathit{init}\ w) \wedge g) \vee \ ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)));\ ((\mathit{init}\ w) \wedge g))$
by (*rule OrChopEqv*)
have *14*: $\vdash (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ \ \ (\mathit{init}\ (\neg\ w)));\ ((\mathit{init}\ w) \wedge g) \longrightarrow$
$\Diamond(\ \ (\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g))$
using *FinChopEqvDiamond*
by (*metis AndFinEqvChopAndEmpty ChopEmpty FiniteChopImpDiamond LeftChopImpChop int-eq*)
have *141*: $\vdash \neg(\ \Diamond(\ \ (\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g))) \longrightarrow$
$\neg\ (\ (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ \ \ (\mathit{init}\ (\neg\ w)));\ ((\mathit{init}\ w) \wedge g))$
using *14* **by** *fastforce*
have *142*: $\vdash ((\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g)) = \#\mathit{False}$
using *Initprop(2)* **by** *fastforce*
have *15*: $\vdash \neg(\ \Diamond(\ \ (\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g)))$
by (*metis 142 NotDiamondAndNot int-simps(21) inteq-reflection*)
have *151*: $\vdash \neg\ (\ (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ \ \ (\mathit{init}\ (\neg\ w)));\ ((\mathit{init}\ w) \wedge g))$
using *15 141* **by** *fastforce*
have *1511*: $\vdash (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)));\ ((\mathit{init}\ w) \wedge g) \longrightarrow \#\mathit{False}$
using *151* **by** (*metis Initprop(2) int-simps(14) inteq-reflection*)
have *152*: $\vdash ((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w));$
$((\mathit{init}\ w) \wedge g) \vee \ (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)));\ ((\mathit{init}\ w) \wedge g) \longrightarrow$
$((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w));\ ((\mathit{init}\ w) \wedge g)$
using *1511* **by** *fastforce*
have *16*: $\vdash (f \wedge \mathit{finite});\ ((\mathit{init}\ w) \wedge g) \longrightarrow ((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w));\ ((\mathit{init}\ w) \wedge g)$
using *12 13 152*
proof −
have $\vdash (f \wedge \mathit{finite});(\mathit{init}\ w \wedge g) \longrightarrow$
$((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ w) \vee (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ \mathit{init}\ w));(\mathit{init}\ w \wedge g)$
by (*metis 12 inteq-reflection lift-and-com*)
then show *?thesis*
using *13 152* **by** *fastforce*
qed
have *17*: $\vdash ((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w));\ ((\mathit{init}\ w) \wedge g) \longrightarrow ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ w));\ g$
by (*rule ChopAndB*)
have *18*: $\vdash (f \wedge \mathit{finite});\ ((\mathit{init}\ w) \wedge g) \longrightarrow ((f \wedge \mathit{finite}) \wedge \ \mathit{fin}\ \ (\mathit{init}\ w));\ g$
using *16 17* **by** *fastforce*
from *10 18* **show** *?thesis* **by** *fastforce*
qed

lemma *DiAndFinEqvChopState*:
$\vdash \quad di\ ((f \wedge finite) \wedge\ fin\ \ (init\ w)) = (f \wedge finite);\ (init\ w)$
proof −
have *1*: $\vdash ((f \wedge finite) \wedge fin(init\ w));\ \#True = (f \wedge finite);((init\ w) \wedge \#True)$
by (*rule AndFinChopEqvStateAndChop*)
have *2*: $\vdash ((init\ w) \wedge \#True) = (init\ w)$
by *auto*
hence *3*: $\vdash ((f \wedge finite);\ ((init\ w) \wedge \#True)) = ((f \wedge finite);\ (init\ w))$
by (*rule RightChopEqvChop*)
have *4*: $\vdash ((f \wedge finite) \wedge\ fin\ \ (init\ w));\ \#True = (f \wedge finite);\ (init\ w)$
using *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *FinNotStateEqvNotFinState*:
$\vdash \quad (\neg(\ fin\ \ \ (init\ w)) \wedge finite\ ) = (fin\ \ \ (init\ (\neg\ w)) \wedge finite\ \ )$
using *FinEqvTrueChopAndEmpty Finprop*(*4*) *Initprop*(*2*) *FiniteImpAnd* **by** (*metis inteq-reflection*)

lemma *BiImpFinEqvYieldsState*:
$\vdash \quad bi\ (f \wedge finite \longrightarrow\ fin\ \ (init\ w)) = (f\ \wedge finite) yields\ \ (init\ w)$
proof −
have *1*: $\vdash\ \ di\ ((f \wedge finite) \wedge fin\ (init\ (\neg\ w))) = (f \wedge finite);\quad (init\ (\neg\ w))$
by (*rule DiAndFinEqvChopState*)
have *2*: $\vdash ((f \wedge finite) \wedge fin(init\ (\neg\ w)))\ \ = ((f \wedge finite) \wedge \neg(fin(init\ w)))$
using *FinNotStateEqvNotFinState* **by** *fastforce*
have *3*: $\vdash ((f \wedge finite) \wedge \neg\ (fin(init\ w))) = (\neg\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w)))$
by *auto*
have *4*: $\vdash ((f \wedge finite) \wedge fin(init\ \ (\neg\ w))) = (\neg\ (f \wedge finite \longrightarrow\ \ fin(init\ w)))$
using *2 3* **by** *fastforce*
hence *5*: $\vdash\ \ di\ ((f \wedge finite) \wedge\ \ fin\ \ \ (init\ (\neg\ w))) =\ \ di\ (\neg\ (f \wedge finite \longrightarrow fin(init\ w)))$
by (*rule DiEqvDi*)
have *6*: $\vdash\ \ di\ (\neg\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w))) = (\neg(\ bi\ (f \wedge finite \longrightarrow fin(init\ w))))$
by (*rule DiNotEqvNotBi*)
have *7*: $\vdash \neg\ (bi\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w))) = (f \wedge finite);(init\ (\neg\ w))$
using *1 5 6 Initprop* **by** *fastforce*
hence *8*: $\vdash\ bi\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w)) = (\neg\ ((f \wedge finite);\ (\neg\ \ (init\ w))))$
by (*metis Initprop*(*2*) *int-eq int-simps*(*7*))
from *8* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *StateImpYields*:
assumes $\vdash \quad (init\ w) \wedge f \wedge finite \longrightarrow\ \ fin\ \ (init\ w1)$
shows $\vdash\ (init\ w) \longrightarrow ((f \wedge finite)\ yields\ \ (init\ w1))$
proof −
have *1*: $\vdash\ (init\ w) \wedge f \wedge finite \longrightarrow\ \ fin\ \ (init\ w1)$
using *assms* **by** *auto*
hence *2*: $\vdash\ (init\ w) \longrightarrow (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w1))$
by *auto*
hence *3*: $\vdash\ (init\ w) \longrightarrow\ bi\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w1))$
by (*rule StateImpBiGen*)

have *4*: $\vdash$ *bi* (*f* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)) = (*f* $\wedge$ *finite*)*yields* (*init w1*)
by (*rule BiImpFinEqvYieldsState*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndYieldsImpYields*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1*
shows $\vdash$ (*init w*) $\wedge$ (*f1 yields g*) $\longrightarrow$ (*f yields g*)
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) $\wedge$ (*f*; ($\neg$ *g*)) $\longrightarrow$ *f1*; ($\neg$ *g*) **by** (*rule StateAndChopImpChopRule*)
hence *3*: $\vdash$ (*init w*) $\wedge$ $\neg$ (*f1*; ($\neg$ *g*)) $\longrightarrow$ $\neg$ (*f*; ($\neg$ *g*)) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *AndYieldsA*:
$\vdash$ *f yields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *yields g*
proof −
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftYieldsImpYields*)
qed

lemma *AndYieldsB*:
$\vdash$ *f1 yields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *yields g*
proof −
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f1* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftYieldsImpYields*)
qed

lemma *RightYieldsImpYields*:
assumes $\vdash$ *g*$\longrightarrow$ *g1*
shows $\vdash$ (*f yields g*) $\longrightarrow$ (*f yields g1*)
proof −
have *1*: $\vdash$ *g* $\longrightarrow$ *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ $\neg$ *g1* $\longrightarrow$ $\neg$ *g* **by** *auto*
hence *3*: $\vdash$ *f*; ($\neg$ *g1*) $\longrightarrow$ *f*; ($\neg$ *g*) **by** (*rule RightChopImpChop*)
hence *4*: $\vdash$ $\neg$ (*f*; ($\neg$ *g*)) $\longrightarrow$ $\neg$ (*f*; ($\neg$ *g1*)) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *RightYieldsEqvYields*:
assumes $\vdash$ *g* = *g1*
shows $\vdash$ (*f yields g*) = (*f yields g1*)
proof −
have *1*: $\vdash$ *g* = *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ ($\neg$ *g*) = ($\neg$ *g1*) **by** *auto*
hence *3*: $\vdash$ *f*; ($\neg$ *g*) = *f*; ($\neg$ *g1*) **by** (*rule RightChopEqvChop*)
hence *4*: $\vdash$ ($\neg$ (*f*; ($\neg$ *g*))) = ($\neg$ (*f*; ($\neg$ *g1*))) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *BoxImpYields*:
$\vdash \quad \Box\ g \longrightarrow (f \wedge \mathit{finite})\ \mathit{yields}\ \ g$
proof −
have *1*: $\vdash (f \wedge \mathit{finite}); (\neg\ \ g) \longrightarrow \Diamond(\neg\ \ g)$ **by** (*rule FiniteChopImpDiamond*)
hence *2*: $\vdash \neg\ (\Diamond(\neg\ \ g)) \longrightarrow \neg\ ((f \wedge \mathit{finite}); (\neg\ \ g))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def always-d-def*)
qed

lemma *BoxEqvFiniteYields*:
$\vdash \quad \Box\ f = \mathit{finite}\ \mathit{yields}\ \ f$
proof −
have *1*: $\vdash \mathit{finite}; (\neg\ \ f) = \Diamond\ (\neg\ \ f)$ **by** (*rule FiniteChopEqvDiamond*)
hence *2*: $\vdash (\neg\ (\mathit{finite}; (\neg\ \ f))) = (\neg(\ \Diamond\ (\neg\ \ f)))$ **by** *auto*
have *3*: $\vdash \Box\ f = (\neg\ (\ \Diamond\ (\neg\ \ f)))$ **by** (*simp add*: *always-d-def*)
have *4*: $\vdash \Box\ f = (\neg\ (\mathit{finite}; (\neg\ \ f)))$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *YieldsGen*:
assumes $\vdash \quad g$
shows $\quad \vdash (f \wedge \mathit{finite})\ \mathit{yields}\ \ g$
proof −
have *1*: $\vdash g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ g$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box\ g \longrightarrow (f \wedge \mathit{finite})\ \mathit{yields}\ \ g$ **by** (*rule BoxImpYields*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *YieldsAndYieldsEqvYieldsAnd*:
$\vdash \quad ((f\ \mathit{yields}\ \ g) \wedge (f\ \ \mathit{yields}\ \ g1)) = f\ \mathit{yields}\ (g \wedge g1)$
proof −
have *1*: $\vdash f; (\neg\ \ g \vee\ \neg\ \ g1) = ((f; (\neg\ \ g)) \vee\ \ (f; (\neg\ \ g1)))$
by (*rule ChopOrEqv*)
hence *2*: $\vdash ((f; (\neg\ \ g)) \vee\ \ (f; (\neg\ \ g1))) = f; (\neg\ \ g \vee\ \ \neg\ \ g1)$
by *auto*
have *3*: $\vdash (\neg\ \ g \vee\ \ \neg\ \ g1)\ \ =\ \ (\neg\ (g \wedge g1))$
by *auto*
hence *4*: $\vdash f; (\neg\ \ g \vee\ \ \neg\ \ g1)\ \ =\ \ f; (\neg\ (g \wedge g1))$
by (*rule RightChopEqvChop*)
have *5*: $\vdash (f; (\neg\ \ g)) \vee\ \ (f; (\neg\ \ g1)) = f; (\neg\ (g \wedge g1))$
using *2 4* **by** *fastforce*
hence *6*: $\vdash (\neg\ (f; (\neg\ \ g)) \wedge \neg\ (f; (\neg\ \ g1))) = (\neg\ (f; (\neg\ (g \wedge g1))))$
by (*metis 1 3 int-simps*(*14*) *int-simps*(*33*) *inteq-reflection*)
from *6* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *YieldsAndYieldsImpAndYieldsAnd*:
$\vdash \quad (f\ \mathit{yields}\ \ g) \wedge (f1\ \mathit{yields}\ \ g1) \longrightarrow (f \wedge f1)\ \mathit{yields}\ (g \wedge g1)$
proof −

have *1*: $\vdash f$ *yields* $g \longrightarrow (f \wedge f1)$ *yields* g
by (*rule AndYieldsA*)
have *2*: $\vdash f1$ *yields* $g1 \longrightarrow (f \wedge f1)$ *yields* $g1$
by (*rule AndYieldsB*)
have *3*: $\vdash ((f \wedge f1)$ *yields* $g \wedge (f \wedge f1)$ *yields* $g1) = (f \wedge f1)$ *yields* $(g \wedge g1)$
by (*rule YieldsAndYieldsEqvYieldsAnd*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *YieldsYieldsEqvChopYields*:
$\vdash$ f *yields* $(g$ *yields* $h) = (f;\ g)$ *yields* h
proof −
have *1*: $\vdash f;\ (g;\ (\neg\ h)) = (f;\ g);\ (\neg\ h)$ **by** (*rule ChopAssoc*)
hence *2*: $\vdash f;\ (g;\ (\neg\ h)) = (f;\ g);\ (\neg\ h)$ **by** *auto*
have *3*: $\vdash g;\ (\neg\ h) = (\neg\ \neg\ (g;\ (\neg\ h)))$ **by** *auto*
hence *4*: $\vdash f;\ (g;\ (\neg\ h)) = f;\ (\neg\ \neg\ (g;\ (\neg\ h)))$ **by** (*rule RightChopEqvChop*)
have *5*: $\vdash f;\ (\neg\ \neg\ (g;\ (\neg\ h))) = (f;\ g);\ (\neg\ h)$ **using** *2 4* **by** *auto*
hence *6*: $\vdash f;\ (\neg\ (g$ *yields* $h)) = (f;\ g);\ (\neg\ h)$ **by** (*simp add*: *yields-d-def*)
hence *7*: $\vdash (\neg\ (f;\ (\neg\ (g$ *yields* $h)))) = (\neg\ ((f;\ g);\ (\neg\ h)))$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *EmptyYields*:
$\vdash$ *empty* *yields* $f = f$
proof −
have *1*: $\vdash$ *empty* ; $(\neg\ f) = (\neg\ f)$ **by** (*rule EmptyChop*)
hence *2*: $\vdash (\neg\ (\ empty\ ;\ (\neg\ f))) = f$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *NextYields*:
$\vdash$ $(\bigcirc f)$ *yields* $g =$ *wnext* $(f$ *yields* $g)$
proof −
have *1*: $\vdash (\bigcirc f);\ (\neg\ g) = \bigcirc(f;\ (\neg\ g))$ **by** (*rule NextChop*)
hence *2*: $\vdash (\neg\ ((\bigcirc f);\ (\neg\ g))) = (\neg\ (\bigcirc(f;\ (\neg\ g))))$ **by** *auto*
hence *3*: $\vdash (\bigcirc f)$ *yields* $g = (\neg\ (\bigcirc(f;\ (\neg\ g))))$ **by** (*simp add*: *yields-d-def*)
have *4*: $\vdash (\neg(\ \bigcirc(f;\ (\neg\ g)))) =$ *wnext* $(\neg\ (f;\ (\neg\ g)))$ **by** (*auto simp*: *wnext-d-def*)
have *5*: $\vdash (\bigcirc f)$ *yields* $g =$ *wnext* $(\neg\ (f;\ (\neg\ g)))$ **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *SkipChopEqvNext*:
$\vdash$ *skip* ; $f = \bigcirc f$
by (*simp add*: *next-d-def*)

lemma *SkipYieldsEqvWeakNext*:
$\vdash$ *skip* *yields* $f =$ *wnext* f
proof −
have *1*: $\vdash$ *skip* ; $(\neg\ f) = \bigcirc(\neg\ f)$ **by** (*rule SkipChopEqvNext*)
hence *2*: $\vdash (\neg\ (\ skip\ ;\ (\neg\ f))) = (\neg(\ \bigcirc(\neg\ f)))$ **by** *auto*

have *3*: $\vdash (\neg\ (\bigcirc(\neg\ f))) = wnext\ f$ **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash (\neg\ (\ skip\ ;\ (\neg\ f))) = wnext\ f$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *NextImpSkipYields*:
$\vdash\ \bigcirc f \longrightarrow skip\ yields\ f$
proof −
have *1*: $\vdash \bigcirc f \longrightarrow wnext\ f$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*
have *2*: $\vdash\ skip\ yields\ f = wnext\ f$ **by** (*rule SkipYieldsEqvWeakNext*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreEqvSkipChopTrue*:
$\vdash\ more\ =\ skip\ ;\ \#True$
proof −
have *1*: $\vdash\ skip\ ;\ \#True = \bigcirc\#True$ **by** (*rule SkipChopEqvNext*)
hence *2*: $\vdash \bigcirc\#True =\ skip\ ;\ \#True$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *more-d-def*)
qed

lemma *MoreChopImpMore*:
$\vdash\ more\ ;\ f \longrightarrow more$
proof −
have *1*: $\vdash (\bigcirc\#True);\ f = \bigcirc(\#True;\ f)$ **by** (*rule NextChop*)
have *2*: $\vdash \bigcirc(\#True;\ f) \longrightarrow more$ **by** (*simp add*: *NextImpNext more-d-def*)
have *3*: $\vdash (\bigcirc\#True;\ f) \longrightarrow more$ **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*metis more-d-def*)
qed

lemma *FmoreChopImpFmore*:
$\vdash\ fmore\ ;\ (f \wedge finite) \longrightarrow fmore$
proof −
have *1*: $\vdash fmore;\ (f \wedge finite) = \bigcirc(finite;\ (f \wedge finite))$
using *FmoreEqvSkipChopFinite* **by** (*metis NextChop inteq-reflection next-d-def*)
have *2*: $\vdash \bigcirc(finite;\ (f \wedge finite)) \longrightarrow fmore$
by (*metis ChopAndB FiniteChopFiniteEqvFinite FmoreEqvSkipChopFinite RightChopImpChop inteq-reflection next-d-def*)
have *3*: $\vdash (\bigcirc finite;\ (f \wedge finite)) \longrightarrow fmore$ **using** *1 2*
by (*metis FmoreEqvSkipChopFinite inteq-reflection next-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopMoreImpMore*:
$\vdash\ f;\ more \longrightarrow more$
proof −
have *1*: $\vdash (f \wedge finite)\ ;\ more \longrightarrow \Diamond\ more$
by (*rule FiniteChopImpDiamond*)
have *11*: $\vdash (f \wedge inf)\ ;\ more \longrightarrow more$
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf Prop11 Prop12 lift-imp-trans*)

have *2*: $\vdash \Diamond\ more \longrightarrow more$
by (*metis FiniteChopMoreEqvMore NowImpDiamond inteq-reflection sometimes-d-def*)
have *3*: $\vdash (f \wedge finite)\ ;\ more \longrightarrow more$
using *1 2* **by** *fastforce*
have *4*: $\vdash f = (\ (f \wedge finite) \vee (f \wedge inf))$
by (*simp add*: *OrFiniteInf*)
hence *5*: $\vdash f;more = (\ (f \wedge finite);more \vee (f \wedge inf);more)$
by (*simp add*: *OrChopEqvRule*)
from *11 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreChopEqvNextDiamond*:
$\vdash \quad fmore\ ;\ f = \bigcirc(\Diamond\ f)$
proof −
have *1*: $\vdash\ fmore\ ;\ f = (\bigcirc\ finite);\ f$
by (*simp add*: *FmoreEqvSkipChopFinite LeftChopEqvChop next-d-def*)
have *2*: $\vdash (\bigcirc\ finite);\ f = \bigcirc(finite;\ f)$
by (*rule NextChop*)
have *3*: $\vdash\ fmore\ ;\ f = \bigcirc(finite;\ f)$
using *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *WeakNextBoxImpMoreYields*:
$\vdash \quad fmore\ \ yields\ \ f = wnext(\ \Box\ f)$
proof −
have *1*: $\vdash fmore\ ;\ (\neg\ f) = \bigcirc(\Diamond\ (\neg f))$ **by** (*rule MoreChopEqvNextDiamond*)
have *2*: $\vdash \bigcirc(\Diamond\ (\neg f)) = \bigcirc(\neg(\Box f))$ **by** (*auto simp*: *always-d-def*)
have *3*: $\vdash \bigcirc(\neg(\Box f)) = (\neg\ (\ wnext(\ \Box\ f)\ ))$ **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash fmore\ ;\ (\neg\ f)\ = (\neg(fmore\ \ yields\ \ f))$ **by** (*simp add*: *yields-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotEqvYieldsMore*:
$\vdash \quad (\neg\ \ f) = f\ yields\ \ more$
proof −
have *1*: $\vdash f;\ \ empty\ = f$ **by** (*rule ChopEmpty*)
hence *2*: $\vdash (\neg\ (f;\ \ empty\ )) = (\neg\ \ f)$ **by** *auto*
have *3*: $\vdash\ \ empty\ = (\neg\ \ more)$ **by** (*auto simp*: *empty-d-def*)
hence *4*: $\vdash f;\ \ empty\ = f;\ (\neg\ \ more)$ **by** (*rule RightChopEqvChop*)
hence *5*: $\vdash (\neg\ (f;\ \ empty\ )) = (\neg\ (f;\ (\neg\ \ more\ )))$ **by** *auto*
have *6*: $\vdash (\neg\ \ f) = (\neg\ (f;\ (\neg\ \ more)\ ))$ **using** *2 5* **by** *fastforce*
from *6* **show** *?thesis* **by** (*metis yields-d-def*)
qed

lemma *LeftChopImpMoreRule*:
assumes $\vdash f \longrightarrow\ \ more$
shows $\vdash f;\ g \longrightarrow\ \ more$
proof −
have *1*: $\vdash f \longrightarrow\ \ more$ **using** *assms* **by** *auto*

hence *2*: $\vdash f;\ g \longrightarrow\ more\ ;\ g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash\ more\ ;\ g \longrightarrow\ more$ **by** (*rule MoreChopImpMore*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *LeftChopImpFMoreRule*:
assumes $\vdash f \longrightarrow\ fmore$
shows $\vdash f;\ (g \wedge finite) \longrightarrow\ fmore$
proof −
have *1*: $\vdash f \longrightarrow\ fmore$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ (g \wedge finite) \longrightarrow\ fmore\ ;\ (g \wedge finite)$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash\ fmore\ ;\ (g \wedge finite) \longrightarrow\ fmore$ **using** *FmoreChopImpFmore* **by** *fastforce*
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *RightChopImpMoreRule*:
assumes $\vdash\ \ g \longrightarrow\ more$
shows $\vdash f;\ g \longrightarrow\ more$
proof −
have *1*: $\vdash g \longrightarrow\ more$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ g \longrightarrow f;\ \ more$ **by** (*rule RightChopImpChop*)
have *3*: $\vdash f;\ \ more\ \longrightarrow\ more$ **by** (*rule ChopMoreImpMore*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NotDiEqvBiNot*:
$\vdash\ \ (\neg\ (\ di\ \ f)) = bi\ (\neg\ \ f)$
proof −
have *1*: $\vdash f = (\neg\ \neg\ \ f)$ **by** *auto*
hence *2*: $\vdash\ \ di\ \ f =\ \ di\ (\neg\ \neg\ \ f)$ **by** (*rule DiEqvDi*)
hence *3*: $\vdash (\neg\ \ (\ di\ \ f)) = (\neg\ (\ di\ (\neg\ \neg\ \ f)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *ChopImpDi*:
$\vdash\ \ f;\ g \longrightarrow\ \ di\ \ f$
proof −
have *1*: $\vdash g \longrightarrow \#True$ **by** *auto*
hence *2*: $\vdash f;\ g \longrightarrow f;\ \#True$ **by** (*rule RightChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *TrueEqvTrueChopTrue*:
$\vdash\ \ \#True = \#True;\ \#True$
proof −
have *1*: $\vdash \#True;\ \#True \longrightarrow \#True$ **by** *auto*
have *2*: $\vdash \#True \longrightarrow\ \ di\ \#True$ **by** (*rule DiIntro*)
hence *3*: $\vdash \#True \longrightarrow \#True;\ \#True$ **by** (*simp add*: *di-d-def*)
from *1 3* **show** *?thesis* **by** *auto*
qed

lemma *DiEqvDiDi*:
$\vdash \quad di\ \ f = \ \ di\ (\ \ di\ \ f)$
proof −
have *1*: $\vdash \#True = \#True; \#True$ **by** (*rule TrueEqvTrueChopTrue*)
hence *2*: $\vdash f; \#True = f; (\#True; \#True)$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f; (\#True; \#True) = (f; \#True); \#True$ **by** (*rule ChopAssoc*)
have *4*: $\vdash f; \#True = (f; \#True); \#True$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis di-d-def*)
qed

lemma *BiEqvBiBi*:
$\vdash \quad bi\ \ f = bi(\ bi\ \ f)$
proof −
have *1*: $\vdash \ \ di\ (\neg\ \ f) = \ \ di(\ \ di\ (\neg\ \ f))$ **by** (*rule DiEqvDiDi*)
have *2*: $\vdash \ \ di\ (\neg\ \ f) = (\neg\ (\ bi\ \ f))$ **by** (*rule DiNotEqvNotBi*)
hence *3*: $\vdash \ \ di\ (\ di\ (\neg\ \ f)) = \ \ di\ (\neg\ (\ bi\ \ f))$ **by** (*rule DiEqvDi*)
have *4*: $\vdash \ \ di\ (\neg\ \ f) = \ \ di\ (\neg(\ \ bi\ \ f))$ **using** *1 3* **by** *fastforce*
hence *5*: $\vdash (\neg\ (\ di\ (\neg\ \ f))) = (\neg\ (\ di\ (\neg\ (\ bi\ \ f))))$ **by** *fastforce*
from *5* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *DiOrEqv*:
$\vdash \quad di\ (f \vee \ \ g) = \ \ (di\ \ f \vee \quad di\ \ g)$
proof −
have *1*: $\vdash (f \vee \ \ g); \#True = (f; \#True \vee \ \ g; \#True)$ **by** (*rule OrChopEqv*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiAndA*:
$\vdash \quad di\ (f \wedge g) \longrightarrow \ \ di\ \ f$
proof −
have *1*: $\vdash (f \wedge g); \#True \longrightarrow f; \#True$ **by** (*rule AndChopA*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiAndB*:
$\vdash \quad di\ (f \wedge g) \longrightarrow \ \ di\ \ g$
proof −
have *1*: $\vdash (f \wedge g); \#True \longrightarrow g; \#True$ **by** (*rule AndChopB*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiAndImpAnd*:
$\vdash \quad di\ (f \wedge g) \longrightarrow \ \ di\ \ f \wedge \ \ di\ \ g$
proof −
have *1*: $\vdash \ \ di\ (f \wedge g) \longrightarrow \ \ di\ \ f$ **by** (*rule DiAndA*)
have *2*: $\vdash \ \ di\ (f \wedge g) \longrightarrow \ \ di\ \ g$ **by** (*rule DiAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiSkipEqvMore*:
$\vdash$ *di skip = more*
proof −
have *1*: $\vdash$ *skip* ; $\#True = \bigcirc\#True$ **by** (*rule SkipChopEqvNext*)
have *2*: $\vdash \bigcirc\#True =$ *more* **by** (*auto simp*: *more-d-def*)
have *3*: $\vdash$ *skip* ; $\#True =$ *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiMoreEqvMore*:
$\vdash$ *di more = more*
proof −
have *1*: $\vdash$ *di* $(\bigcirc\ \#True\ ) = \bigcirc($ *di* $\#True)$
by (*rule DiNext*)
have *2*: $\vdash \bigcirc($ *di* $\#True) \longrightarrow$ *more*
by (*metis 1 ChopImpDi TrueEqvTrueChopTrue di-d-def int-eq more-d-def*)
have *3*: $\vdash$ *di*$(\bigcirc\ \#True) \longrightarrow$ *more*
using *1 2* **by** *fastforce*
hence *4*: $\vdash$ *di more* $\longrightarrow$ *more*
by (*simp add*: *more-d-def*)
have *5*: $\vdash$ *more* $\longrightarrow$ *di more*
by (*rule ImpDi*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *DiIfEqvRule*:
assumes $\vdash$ $f =$ *if*$_i$ (*init w*) *then g else h*
shows $\vdash$ *di f* = *if*$_i$ (*init w*) *then* (*di g*) *else* (*di h*)
proof −
have *1*: $\vdash f =$ *if*$_i$ (*init w*) *then g else h* **using** *assms* **by** *auto*
hence *2*: $\vdash f; \#True =$ *if*$_i$ (*init w*) *then* $(g; \#True)$ *else* $(h; \#True)$ **by** (*rule IfChopEqvRule*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiEmpty*:
$\vdash$ *di empty*
proof −
have *1*: $\vdash \#True$ **by** *auto*
have *2*: $\vdash$ *empty* ; $\#True = \#True$ **by** (*rule EmptyChop*)
have *3*: $\vdash$ *empty* ; $\#True$ **using** *1 2* **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DaNotEqvNotBa*:
$\vdash$ *da* $(\neg\ f) = (\neg$ (*ba f*))
proof −
have *1*: $\vdash$ *ba f* $= (\neg$ (*da* $(\neg\ f)))$ **by** (*simp add*: *ba-d-def*)
from *1* **show** *?thesis* **by** *fastforce*
qed

lemma *DaEqvDa*:
assumes $\vdash\ f = g$
shows $\vdash\ da\ f = da\ g$
using *assms* **using** *int-eq* **by** *force*

lemma *DaEqvNotBaNot*:
$\vdash\ da\ f = (\neg\ (\ ba\ (\neg\ f)))$
proof −
have *1*: $\vdash\ ba\ (\neg\ f) = (\neg\ (\ da\ (\neg\ \neg\ f)))$ **by** (*simp add*: *ba-d-def*)
hence *2*: $\vdash\ da\ (\neg\ \neg\ f) = (\neg(\ ba\ (\neg\ f)))$ **by** *fastforce*
have *3*: $\vdash f = (\neg\ \neg\ f)$ **by** *simp*
hence *4*: $\vdash\ da\ f = da\ (\neg\ \neg\ f)$ **by** (*rule DaEqvDa*)
from *2 4* **show** *?thesis* **by** *simp*
qed

lemma *BaElim*:
$\vdash\ ba\ f \longrightarrow f$
proof −
have *1*: $\vdash\ ba\ f = \Box(bi\ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash\ bi\ f \longrightarrow f$ **by** (*rule BiElim*)
hence *3*: $\vdash \Box(bi\ f \longrightarrow f)$ **by** (*rule BoxGen*)
have *4*: $\vdash \Box(bi\ f \longrightarrow f) \longrightarrow \Box(bi\ f) \longrightarrow \Box\ f$ **by** (*rule BoxImpDist*)
have *5*: $\vdash \Box(bi\ f) \longrightarrow \Box\ f$ **using** *3 4 MP* **by** *fastforce*
have *6*: $\vdash \Box\ f \longrightarrow f$ **by** (*rule BoxElim*)
from *1 5 6* **show** *?thesis* **using** *BaImpBt lift-imp-trans* **by** *metis*
qed

lemma *DaIntro*:
$\vdash\ f \longrightarrow\ da\ f$
proof −
have *1*: $\vdash\ ba\ (\neg\ f) \longrightarrow (\neg\ f)$ **by** (*rule BaElim*)
hence *2*: $\vdash \neg\ \neg\ f \longrightarrow \neg\ (\ ba\ (\neg\ f))$ **by** *fastforce*
have *3*: $\vdash f = (\neg\ \neg\ f)$ **by** *simp*
have *4*: $\vdash\ da\ f = (\neg\ (\ ba\ (\neg\ f)))$ **by** (*rule DaEqvNotBaNot*)
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BaGen*:
assumes $\vdash\ f$
shows $\vdash\ ba\ f$
proof −
have *1*: $\vdash\ f$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ f$ **by** (*rule BoxGen*)
hence *3*: $\vdash bi(\ \Box\ f)$ **by** (*rule BiGen*)
have *4*: $\vdash\ ba\ f = bi\ (\Box\ f)$ **by** (*rule BaEqvBiBt*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpDist*:

$\vdash$ $ba\ (f \longrightarrow g) \longrightarrow\ ba\ \ f \longrightarrow\ \ ba\ \ g$
proof −
have *1*: $\vdash$ $bi\ (f \longrightarrow g) \longrightarrow (bi\ \ f \longrightarrow bi\ \ g)$
by (*rule BiImpDist*)
hence *2*: $\vdash \Box(bi\ (f \longrightarrow g) \longrightarrow (bi\ \ f \longrightarrow bi\ \ g))$
by (*rule BoxGen*)
have *3*: $\vdash \Box(bi\ (f \longrightarrow g) \longrightarrow (bi\ \ f \longrightarrow bi\ \ g))$
$\longrightarrow$
$(\Box\ (bi\ (f \longrightarrow g)) \longrightarrow (\Box(bi\ \ f) \longrightarrow \Box(bi\ \ g)))$
by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)
have *4*: $\vdash \Box(bi\ (f \longrightarrow g)) \longrightarrow (\Box(bi\ \ f) \longrightarrow \Box(bi\ \ g))$
using *2 3 MP* **by** *fastforce*
have *5*: $\vdash\ \ ba\ (f \longrightarrow g) = \Box(bi\ (f \longrightarrow g))$
by (*rule BaEqvBtBi*)
have *6*: $\vdash\ \ ba\ \ f = \Box(bi\ \ f)$
by (*rule BaEqvBtBi*)
have *7*: $\vdash\ \ ba\ \ g = \Box(bi\ \ g)$
by (*rule BaEqvBtBi*)
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *BiAndEqv*:
$\vdash$ $bi\ (f \wedge g) = (bi\ f \wedge bi\ g)$
proof −
have *1*: $\vdash$ $di\ (\neg\ f \vee \neg\ g) = (di\ (\neg\ f) \vee\ di\ (\neg\ g))$
by (*simp add*: *DiOrEqv*)
have *2*: $\vdash (\neg\ (di\ (\neg\ f \vee \neg\ g))) = (\neg\ di\ (\neg\ f) \wedge \neg\ di\ (\neg\ g))$
using *1* **by** *auto*
have *3*: $\vdash (f \wedge g) = (\neg\ (\neg\ f \vee \neg\ g))$
by *fastforce*
have *4*: $\vdash$ $bi\ (f \wedge g) = (\neg\ (di\ (\neg\ f \vee \neg\ g)))$
unfolding *bi-d-def* **using** *3* **by** (*metis int-simps(4) inteq-reflection*)
from *2 4* **show** *?thesis* **unfolding** *bi-d-def* **by** (*metis inteq-reflection*)
qed

lemma *BaAndEqv*:
$\vdash$ $\ \ ba\ (f \wedge g) =\ \ (ba\ \ f \wedge\ \ ba\ \ g)$
proof −
have *1*: $\vdash\ \ \ ba\ (f \wedge g) =\ \ \ \Box(bi\ (f \wedge g))$
by (*rule BaEqvBtBi*)
have *2*: $\vdash\ \ bi\ (f \wedge g) = (bi\ f \wedge bi\ g)$
by (*simp add*: *BiAndEqv*)
hence *3*: $\vdash \Box(bi\ (f \wedge g)) = \Box(bi\ f \wedge bi\ g)$
using *BoxEqvBox* **by** *blast*
have *4*: $\vdash \Box(bi\ f \wedge bi\ g) = (\Box(bi\ f)\ \ \wedge \Box(bi\ g))$
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have *5*: $\vdash ba\ \ f = \Box(bi\ \ f)$
by (*rule BaEqvBtBi*)
have *6*: $\vdash\ \ ba\ \ g = \Box(bi\ \ g)$
by (*rule BaEqvBtBi*)

from *1 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaEqvBa*:
$\vdash$ *ba* $(f = g) \longrightarrow$ (*ba* $f =$ *ba* g)
proof −
have *1*: $\vdash$ *ba* $(f \longrightarrow g) \longrightarrow$ *ba* $f \longrightarrow$ *ba* g **by** (*rule BaImpDist*)
have *2*: $\vdash$ *ba* $(g \longrightarrow f) \longrightarrow$ *ba* $g \longrightarrow$ *ba* f **by** (*rule BaImpDist*)
have *25*: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *fastforce*
have *3*: $\vdash$ *ba* $(f = g) =$ *ba* $((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (*metis 25 BaAndEqv inteq-reflection*)
have *4*: $\vdash$ *ba* $((f \longrightarrow g) \wedge (g \longrightarrow f)) = (ba((f \longrightarrow g)) \wedge ba((g \longrightarrow f)))$ **by** (*rule BaAndEqv*)
have *5*: $\vdash$ $((ba\ f \longrightarrow ba\ g) \wedge (ba\ g \longrightarrow ba\ f)) = (ba\ f = ba\ g)$ **by** *auto*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBa*:
assumes $\vdash$ $f \longrightarrow g$
shows $\vdash ba\ f \longrightarrow ba\ g$
using *BaGen BaImpDist MP assms* **by** *metis*

lemma *BaEqvBa*:
assumes $\vdash$ $f = g$
shows $\vdash$ *ba* $f =$ *ba* g
using *BaGen BaImpBaEqvBa MP assms* **by** *metis*

lemma *DaImpDa*:
assumes $\vdash$ $f \longrightarrow g$
shows $\vdash$ *da* $f \longrightarrow$ *da* g
using *assms* **by** (*metis DaEqvDtDi DiAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *DiamondEqvDiamondDiamond*:
$\vdash \Diamond f = \Diamond (\Diamond f)$
proof −
have *1*: $\vdash \Diamond (\Diamond f) =$ *finite*;(*finite*;*f*)
by (*simp add*: *sometimes-d-def*)
have *2*: $\vdash$ *finite*;(*finite*;*f*) = (*finite*;*finite*);*f*
by (*rule ChopAssoc*)
have *3*: $\vdash$ (*finite*;*finite*);*f* = *finite*;*f*
by (*simp add*: *LeftChopEqvChop FiniteChopFiniteEqvFinite*)
have *4*: $\vdash$ *finite*;*f* = $\Diamond$*f*
by (*simp add*: *sometimes-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DaEqvDaDa*:
$\vdash$ *da* $f =$ *da* (*da* f)
proof −
have *1*: $\vdash$ *da* $f = \Diamond$(*di* f)
by (*rule DaEqvDtDi*)
have *2*: $\vdash$ *di* $f =$ (*di* (*di* f))

by (*rule DiEqvDiDi*)
hence *3*: $\vdash \Diamond\ (\ di\ \ f) = \Diamond\ (di\ \ (di\ \ f))$
by (*rule DiamondEqvDiamond*)
have *4*: $\vdash \Diamond\ \ (di\ \ f) = \Diamond(\Diamond\ (di\ \ (di\ \ f)))$
using *DiamondEqvDiamondDiamond DiEqvDiDi* **using** *3* **by** *fastforce*
have *5*: $\vdash \Diamond\ \ (di\ \ (di\ \ f)) =\ \ di\ (\Diamond\ \ (di\ \ f))$
by (*rule DtDiEqvDiDt*)
hence *6*: $\vdash \Diamond(\Diamond\ \ (di\ \ (di\ \ f))) = \Diamond\ (di\ (\Diamond\ \ (di\ \ f)))$
by (*rule DiamondEqvDiamond*)
have *7*: $\vdash\ \ da\ \ f = \Diamond\ (di(\ \Diamond\ (\ di\ \ f)))$
using *1 3 4 6* **by** *fastforce*
have *8*: $\vdash\ \ da\ (\Diamond\ (\ di\ \ f)) = \Diamond(\ di\ (\Diamond\ \ (di\ \ f)))$
by (*rule DaEqvDtDi*)
have *9*: $\vdash\ \ da\ (\ da\ \ f) =\ \ da\ (\Diamond\ \ (di\ \ f))$
using *1* **by** (*rule DaEqvDa*)
from *7 8 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BaEqvBaBa*:
$\vdash\ \ \ ba\ \ f =\ \ ba\ \ (ba\ \ f)$
proof −
have *1*: $\vdash\ \ da\ (\neg\ \ f) =\ \ da\ \ (da\ (\neg\ \ f))$ **by** (*rule DaEqvDaDa*)
have *2*: $\vdash\ \ da\ \ (da\ (\neg\ \ f)) = (\neg\ \ (ba\ (\neg\ \ (da\ (\neg\ \ f)))))$ **by** (*rule DaEqvNotBaNot*)
have *3*: $\vdash (\neg\ \ (da\ \ (da\ (\neg\ \ f)))) =\ \ ba\ (\neg\ \ (da\ (\neg\ \ f)))$ **by** (*auto simp*: *ba-d-def*)
have *4*: $\vdash (\neg\ \ (da\ (\neg\ \ f))) =\ \ ba\ (\neg\ \ (da\ (\neg\ \ f)))$ **using** *1 2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis ba-d-def*)
qed

lemma *BaLeftChopImpChop*:
$\vdash\ \ \ ba\ (f \longrightarrow f1) \longrightarrow f;\ g \longrightarrow f1;\ g$
proof −
have *1*: $\vdash\ \ ba\ (f \longrightarrow f1) \longrightarrow bi\ (f \longrightarrow f1)$ **by** (*rule BaImpBi*)
have *2*: $\vdash bi\ (f \longrightarrow f1) \longrightarrow f;\ g \longrightarrow f1;\ g$ **by** (*rule BiChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaRightChopImpChop*:
$\vdash\ \ \ ba\ (g \longrightarrow g1) \longrightarrow f;\ g \longrightarrow f;\ g1$
proof −
have *1*: $\vdash\ \ ba\ (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (*rule BaImpBt*)
have *2*: $\vdash \Box(g \longrightarrow g1) \longrightarrow f;\ g \longrightarrow f;\ g1$ **by** (*rule BoxChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopAndBaImport*:
$\vdash\ \ (f;\ f1) \wedge\ \ ba\ \ g \longrightarrow (f \wedge g);\ (f1 \wedge g)$
proof −
have *1*: $\vdash\ \ ba\ \ g \wedge (f;\ f1) \longrightarrow (g \wedge f);\ (g \wedge f1)$ **by** (*rule BaAndChopImport*)
have *2*: $\vdash (g \wedge f);\ (g \wedge f1) = (f \wedge g);\ (f1 \wedge g)$ **by** (*rule AndChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*

qed

lemma *BaAndChopImportA*:
$\vdash$ *ba f* $\wedge$ *g*;*g1* $\longrightarrow$ (*f* $\wedge$ *g*);*g1*
by (*meson BaAndChopImport ChopAndB lift-imp-trans*)

lemma *BaAndChopImportB*:
$\vdash$ *ba f* $\wedge$ *g*;*g1* $\longrightarrow$ (*f* $\wedge$ *g*);(*ba f* $\wedge$ *g1*)
proof −
have *1*: $\vdash$ *ba f* = *ba* (*ba f*)
by (*simp add*: *BaEqvBaBa*)
have *2*: $\vdash$ *ba* (*ba f*) $\wedge$ *g*;*g1* $\longrightarrow$ *g*;(*ba f* $\wedge$ *g1*)
by (*metis AndChopB BaAndChopImport lift-imp-trans*)
have *3*: $\vdash$ *ba f* $\wedge$ *g*;(*ba f* $\wedge$ *g1*) $\longrightarrow$ (*f* $\wedge$ *g*);(*ba f* $\wedge$ *g1*)
by (*simp add*: *BaAndChopImportA*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaImpBaAnd*:
$\vdash$ *ba h* $\longrightarrow$ *ba*(*g* $\longrightarrow$ *ba h* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *ba h* $\longrightarrow$ (*g* $\longrightarrow$ *ba h* $\wedge$ *g*) **by** *fastforce*
hence *2*: $\vdash$ *ba*(*ba h*) $\longrightarrow$ *ba*(*g* $\longrightarrow$ *ba h* $\wedge$ *g*) **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba h* = *ba*(*ba h*) **by** (*rule BaEqvBaBa*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaChopImpChopBa*:
$\vdash$ *ba f* $\longrightarrow$ *g*; *g1* $\longrightarrow$ *g*; ((*ba f*) $\wedge$ *g1*)
proof −
have *1*: $\vdash$ *ba f* $\longrightarrow$ *ba* (*g1* $\longrightarrow$ (*ba f*) $\wedge$ *g1*) **by** (*rule BaImpBaImpBaAnd*)
have *2*: $\vdash$ *ba* (*g1* $\longrightarrow$ *ba f* $\wedge$ *g1*) $\longrightarrow$ *g*; *g1* $\longrightarrow$ *g*; (*ba f* $\wedge$ *g1*) **by** (*rule BaRightChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiNotBaImpNotBa*:
$\vdash$ *di* (¬ (*ba f*)) $\longrightarrow$ ¬ (*ba f*)
proof −
have *1*: $\vdash$ *ba f* = *ba*(*ba f*) **by** (*rule BaEqvBaBa*)
have *2*: $\vdash$ *ba* (*ba f*) $\longrightarrow$ *bi* (*ba f*) **by** (*rule BaImpBi*)
have *3*: $\vdash$ *ba f* $\longrightarrow$ *bi* (*ba f*) **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *ba f* $\longrightarrow$ ¬ (*di* (¬ (*ba f*))) **by** (*simp add*: *bi-d-def*)
from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBaChopImpNotBa*:
$\vdash$ (¬ (*ba f*)); *g* $\longrightarrow$ ¬ (*ba f*)
proof −
have *1*: $\vdash$ (¬ (*ba f*)); *g* $\longrightarrow$ *di* (¬ (*ba f*)) **by** (*rule ChopImpDi*)
have *2*: $\vdash$ *di* (¬ (*ba f*)) $\longrightarrow$ ¬ (*ba f*) **by** (*rule DiNotBaImpNotBa*)

from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DiamondFinImpFin*:
$\vdash \Diamond\ (fin\ f) \longrightarrow fin\ f$
proof −
have *1*: $\vdash fin\ f = \#True;(f \wedge empty)$
by (*rule FinEqvTrueChopAndEmpty*)
hence *2*: $\vdash \Diamond\ (fin\ f) = finite;(\#True;(f \wedge empty))$
by (*metis FiniteChopFiniteEqvFinite LeftChopEqvChop inteq-reflection sometimes-d-def*)
have *3*: $\vdash finite;(\#True;(f \wedge empty)) = (finite;\#True);(f \wedge empty)$
by (*rule ChopAssoc*)
have *4*: $\vdash (finite;\#True);(f \wedge empty) \longrightarrow \#True;(f \wedge empty)$
using *1 2 3 DiamondFin* **by** *fastforce*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopFinImpFin*:
$\vdash\ (f \wedge finite);\ fin\ (init\ w) \longrightarrow\ fin\ (init\ w)$
proof −
have *1*: $\vdash (f \wedge finite);\ fin\ (init\ w) \longrightarrow \Diamond\ (\ fin\ (init\ w))$ **by** (*rule FiniteChopImpDiamond*)
have *2*: $\vdash \Diamond\ (fin\ (init\ w)) \longrightarrow\ fin\ (init\ w)$ **by** (*rule DiamondFinImpFin*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FiniteRightChopEqvChop*:
assumes $\vdash finite \longrightarrow g = g1$
shows $\vdash finite \longrightarrow f;g = f;g1$
using *assms* **by** (*auto simp add*: *Valid-def itl-defs*)

lemma *FinImpYieldsFin*:
$\vdash\ \ fin\ (init\ w) \wedge finite \longrightarrow (f \wedge finite)\ yields\ (fin\ (init\ w) \wedge finite)$
proof −
have *1*: $\vdash (f \wedge finite);\ (fin\ \ (init\ (\neg\ w)) \wedge finite) \longrightarrow\ (fin\ \ (init\ (\neg\ w)) \wedge finite)$
by (*metis* (*no-types*, *lifting*) *ChopAndB FiniteChopEqvDiamond FiniteChopFinExportA*
FiniteChopFiniteEqvFinite FiniteChopImpDiamond Prop12 inteq-reflection
lift-and-com lift-imp-trans)
have *2*: $\vdash finite \longrightarrow (\neg\ (\ fin\ (init\ w) \wedge finite)) = (fin\ (init\ (\neg\ w)) \wedge finite)$
using *FinNotStateEqvNotFinState* **by** *fastforce*
hence *3*: $\vdash\ finite \longrightarrow (f \wedge finite);\ (\neg\ (\ fin\ (init\ w) \wedge finite)) =$
$(f \wedge finite);\ (\ fin\ (init\ (\neg\ w)) \wedge finite)$
using *FiniteRightChopEqvChop*[*of LIFT*$(\neg\ (\ fin\ (init\ w) \wedge finite))$
LIFT$(fin\ (init\ (\neg\ w)) \wedge finite)$ *LIFT*$(f \wedge finite)$]
by *blast*
have *4*: $\vdash (f \wedge finite);\ (\neg\ (\ fin\ \ (init\ w) \wedge finite)) \longrightarrow (\neg\ (\ fin\ \ (init\ w) \wedge finite))$
using *1 2 3* **by** *fastforce*
hence *5*: $\vdash\ fin\ \ (init\ w) \wedge finite \longrightarrow \neg\ ((f \wedge finite);\ (\neg\ (\ fin\ \ (init\ w) \wedge finite)))$
by *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ChopAndFin*:
$\vdash ((f;\ g) \wedge\ (fin\ \ (init\ w) \wedge finite)) = (f \wedge finite);\ (g \wedge\ \ (fin\ \ (init\ w) \wedge finite))$
proof −
have *1*: $\vdash\ fin\ \ (init\ w) \wedge finite \longrightarrow (f \wedge finite)\ yields\ (\ fin\ \ (init\ w) \wedge finite)$
by (*rule FinImpYieldsFin*)
have *10*: $\vdash ((f;\ g) \wedge\ \ (fin\ \ (init\ w) \wedge finite)) =$
$(((f \wedge finite);\ (g \wedge finite)) \wedge\ \ fin\ \ (init\ w) \wedge finite)$
using *ChopAndFiniteDist*[*of f g*] **by** *auto*
have *2*: $\vdash (f;\ g) \wedge\ \ (fin\ \ (init\ w) \wedge finite) \longrightarrow$
$((f \wedge finite);\ (g \wedge finite)) \wedge (f \wedge finite)\ yields\ \ (\ fin\ \ (init\ w) \wedge finite)$
using *1 10* **by** *fastforce*
have *3*: $\vdash ((f \wedge finite);\ (g \wedge finite)) \wedge (f \wedge finite)\ yields\ \ (\ fin\ \ (init\ w) \wedge finite) \longrightarrow$
$(f \wedge finite);\ ((g \wedge finite) \wedge\ \ (fin\ \ (init\ w) \wedge finite))$
using *ChopAndYieldsImp* **by** *blast*
have *30*: $\vdash ((g \wedge finite) \wedge\ \ (fin\ \ (init\ w) \wedge finite)) = (g \wedge fin\ \ (init\ w) \wedge finite)$
by *auto*
have *4*: $\vdash (f;\ g) \wedge\ \ (fin\ \ (init\ w) \wedge finite) \longrightarrow (f \wedge finite);\ (g \wedge\ \ fin\ \ (init\ w) \wedge finite)$
using *2 3 30*
by (*metis* (*mono-tags*, *lifting*) *inteq-reflection lift-imp-trans*)
have *11*: $\vdash (f \wedge finite);\ (g \wedge\ \ fin\ \ (init\ w) \wedge finite) \longrightarrow (f \wedge finite);\ (g \wedge finite)$
using *ChopAndA* **by** (*metis 30 inteq-reflection*)
have *12*: $\vdash (f \wedge finite);\ (g \wedge\ \ fin\ \ (init\ w) \wedge finite) \longrightarrow$
$(f \wedge finite);\ \ (fin\ \ (init\ w) \wedge finite)$
by (*rule ChopAndB*)
have *13*: $\vdash (f \wedge finite);\ \ (fin\ \ (init\ w) \wedge finite) \longrightarrow \Diamond\ (\ fin\ \ (init\ w) \wedge finite)$
using *FiniteChopImpDiamond* **by** *blast*
have *14*: $\vdash \Diamond(\ fin\ \ (init\ w) \wedge finite) \longrightarrow\ fin\ \ (init\ w)$
by (*metis ChopAndA DiamondFin inteq-reflection sometimes-d-def*)
have *15*: $\vdash (f \wedge finite);\ (g \wedge\ \ fin\ \ (init\ w) \wedge finite) \longrightarrow$
$((f \wedge finite);\ (g \wedge finite)) \wedge\ \ fin\ \ (init\ w)$
using *11 12 13 14* **by** *fastforce*
from *4 15* **show** *?thesis* **by** (*metis ChopAndFiniteDist Prop12 int-iffI inteq-reflection*)
qed

lemma *ChopAndNotFin*:
$\vdash\ \ (f;\ g \wedge \neg\ (\ fin\ \ (init\ w)) \wedge finite) = (f \wedge finite);\ (g \wedge \neg\ (\ fin\ \ (init\ w)) \wedge finite)$
proof −
have *1*: $\vdash (f;\ g \wedge\ \ fin\ \ \ (init\ (\neg\ w)) \wedge finite) =$
$(f \wedge finite);\ (g \wedge\ \ fin\ \ \ (init\ (\neg\ w)) \wedge finite)$
by (*rule ChopAndFin*)
have *2*: $\vdash\ \ (fin\ \ \ (init\ (\neg\ w)) \wedge finite) = (\ (\neg\ \ (\ fin\ \ (init\ w)\ )) \wedge finite)$
using *FinNotStateEqvNotFinState* **by** *fastforce*
show *?thesis* **by** (*metis 1 2 int-eq*)
qed

lemma *FinChopChain*:
$\vdash\ \ (((init\ w\ )\ \wedge finite \longrightarrow\ \ fin\ \ (init\ w1)));$
$(((init\ w1)\ \wedge finite \longrightarrow\ \ fin\ \ (init\ w2)))$
$\wedge\ finite$

$\longrightarrow$ (((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)))
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ *finite* $\wedge$
((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)); ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))
$\longrightarrow$
((*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)));
(((*init w1*) $\wedge$ *finite*$\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
using *ChopAndFiniteDist StateAndChopImport*
by (*metis* (*no-types*, *opaque-lifting*) *inteq-reflection lift-and-com*)
have *2*: $\vdash$ (*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin*(*init w1*)) $\longrightarrow$ *fin* (*init w1*) $\wedge$ *finite*
by *auto*
have *3*: $\vdash$ ((*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)));
(((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
$\longrightarrow$
(*fin* (*init w1*) $\wedge$ *finite*); (((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
using *2 LeftChopImpChop* **by** *blast*
have *4*: $\vdash$ (*fin* (*init w1*) $\wedge$ *finite*); (((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*) =
$\Diamond$((*init w1*) $\wedge$ ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
using *FinChopEqvDiamond* **by** *blast*
have *41*: $\vdash$ ((*init w1*) $\wedge$ *finite* $\wedge$ ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))) $\longrightarrow$ *fin* (*init w2*)
by *auto*
have *42*: $\vdash$ $\Diamond$((*init w1*) $\wedge$ *finite* $\wedge$ ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))) $\longrightarrow$ $\Diamond$(*fin* (*init w2*))
using *41 DiamondImpDiamond* **by** *blast*
have *5*: $\vdash$ $\Diamond$(*fin*(*init w2*)) $\longrightarrow$ *fin* (*init w2*)
using *DiamondFinImpFin* **by** *blast*
have *6*: $\vdash$ (*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*));
((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))
$\longrightarrow$ *fin* (*init w2*)
using *1 3 4 5 42*
using *ChopAndCommute FinChopEqvDiamond* **by** *fastforce*
from *6* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRule*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)
$\vdash$ (*init w1*)$\wedge$ *f1* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)
shows $\vdash$ (*init w*) $\wedge$ (*f*; *f1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ (*f*; *f1*) $\wedge$ *finite* $\longrightarrow$ ((*init w*) $\wedge$ *f* $\wedge$ *finite*); (*f1* $\wedge$ *finite*)
using *StateAndChopImport*
by (*metis ChopAndFiniteDist inteq-reflection*)
have *2*: $\vdash$ (*init w*) $\wedge$ *f* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*) $\wedge$ *finite*
using *assms* **by** *auto*
hence *3*: $\vdash$ ((*init w*) $\wedge$ *f* $\wedge$ *finite*); (*f1* $\wedge$ *finite*) $\longrightarrow$ (*fin* (*init w1*) $\wedge$ *finite*); (*f1* $\wedge$ *finite*)
by (*rule LeftChopImpChop*)
have *4*: $\vdash$ (*fin* (*init w1*) $\wedge$ *finite*); (*f1* $\wedge$ *finite*) = $\Diamond$((*init w1*) $\wedge$ *f1* $\wedge$ *finite*)
by (*rule FinChopEqvDiamond*)
have *5*: $\vdash$ (*init w1*) $\wedge$ *f1* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)
using *assms* **by** *auto*
hence *6*: $\vdash$ $\Diamond$((*init w1*) $\wedge$ *f1* $\wedge$ *finite*) $\longrightarrow$ $\Diamond$ (*fin* (*init w2*))

by (*rule DiamondImpDiamond*)
have *7*: $\vdash \Diamond($ *fin* $($*init w2*$)) \longrightarrow$ *fin* $($*init w2*$)$
using *DiamondFinImpFin* **by** *blast*
from *1 3 4 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRep*:
assumes $\vdash$ $($*init w*$) \wedge f \wedge$ *finite* $\longrightarrow$ *f1* $\wedge$ *fin* $($*init w1*$)$
$\vdash$ $($*init w1*$) \wedge g \wedge$ *finite* $\longrightarrow$ *g1*
shows $\vdash$ $($*init w*$) \wedge (f; g) \wedge$ *finite* $\longrightarrow ((f1 \wedge$ *finite*$); g1)$
proof −
have *1*: $\vdash ($*init w*$) \wedge f \wedge$ *finite* $\longrightarrow ((f1 \wedge$ *finite*$) \wedge$ *fin* $($*init w1*$))$
using *assms* **by** *auto*
hence *2*: $\vdash ($*init w*$) \wedge ((f \wedge$ *finite*$); (g \wedge$ *finite*$)) \longrightarrow$
$((f1 \wedge$ *finite*$) \wedge$ *fin* $($*init w1*$)); (g \wedge$ *finite*$)$
using *StateAndChopImpChopRule* **by** *blast*
have *3*: $\vdash ((f1 \wedge$ *finite*$) \wedge$ *fin* $($*init w1*$)); (g \wedge$ *finite*$) =$
$(f1 \wedge$ *finite*$); (($*init w1*$) \wedge (g \wedge$ *finite*$))$
using *AndFinChopEqvStateAndChop* **by** *blast*
have *4*: $\vdash ($*init w1*$) \wedge g \wedge$ *finite* $\longrightarrow$ *g1*
using *assms* **by** *auto*
hence *5*: $\vdash (f1 \wedge$ *finite*$); (($*init w1*$) \wedge g \wedge$ *finite*$) \longrightarrow (f1 \wedge$ *finite*$);$ *g1*
using *RightChopImpChop* **by** *blast*
from *2 3 5* **show** *?thesis* **using** *ChopAndFiniteDist* **by** *fastforce*
qed

lemma *ChopRepAndFin*:
assumes $\vdash$ $($*init w*$) \wedge f \wedge$ *finite* $\longrightarrow$ *f1* $\wedge$ *fin* $($*init w1*$)$
$\vdash$ $($*init w1*$) \wedge g \wedge$ *finite* $\longrightarrow$ *g1* $\wedge$ *fin* $($*init w2*$)$
shows $\vdash$ $($*init w*$) \wedge (f; g) \wedge$ *finite* $\longrightarrow ((f1 \wedge$ *finite*$);$ *g1*$) \wedge$ *fin* $($*init w2*$)$
proof −
have *1*: $\vdash ($*init w*$) \wedge f \wedge$ *finite* $\longrightarrow$ *f1* $\wedge$ *fin* $($*init w1*$)$
using *assms* **by** *auto*
have *2*: $\vdash ($*init w1*$) \wedge g \wedge$ *finite* $\longrightarrow$ *g1* $\wedge$ *fin* $($*init w2*$)$
using *assms* **by** *auto*
have *3*: $\vdash ($*init w*$) \wedge (f; g) \wedge$ *finite* $\longrightarrow (f1 \wedge$ *finite*$); (g1 \wedge$ *fin* $($*init w2*$))$
using *1 2* **by** (*rule ChopRep*)
have *4*: $\vdash (f1 \wedge$ *finite*$); (g1 \wedge$ *fin* $($*init w2*$)) \longrightarrow (f1 \wedge$ *finite*$);$ *g1*
by (*rule ChopAndA*)
have *5*: $\vdash (f1 \wedge$ *finite*$); (g1 \wedge$ *fin* $($*init w2*$)) \longrightarrow (f1 \wedge$ *finite*$);$ *fin* $($*init w2*$)$
by (*rule ChopAndB*)
have *6*: $\vdash (f1 \wedge$ *finite*$);$ *fin* $($*init w2*$) \longrightarrow$ *fin* $($*init w2*$)$
by (*rule ChopFinImpFin*)
from *1 2 3 4 5 6* **show** *?thesis* **by** (*meson Prop12 lift-imp-trans*)
qed

lemma *TrueChopMoreEqvMore*:
$\vdash$ #*True* ; *more* = *more*
by (*metis ChopMoreImpMore EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteChopMoreEqvMore*
LeftChopImpChop Prop09 int-eq-true int-iffI inteq-reflection)

lemma *FiniteChopFmoreEqvFmore*:
$\vdash$ *finite*;*fmore* = *fmore*
by (*metis TrueChopAndFiniteEqvAndFiniteChopFinite TrueChopMoreEqvMore fmore-d-def inteq-reflection*)

lemma *MoreChopLoop*:
assumes $\vdash$ *f* $\longrightarrow$ *fmore* ; *f*
shows $\vdash$ *finite* $\longrightarrow \neg$ *f*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *fmore* ; *f*
using *assms* **by** *auto*
hence *11*: $\vdash \Diamond$ (*f*) $\longrightarrow \Diamond$ (*fmore*;*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash \Diamond$ (*fmore*;*f*) = *finite*;(*fmore*;*f*)
by (*simp add*: *sometimes-d-def*)
have *13*: $\vdash$ *finite*;(*fmore*;*f*) = (*finite*;*fmore*);*f*
by (*rule ChopAssoc*)
have *14*: $\vdash \Diamond$ (*fmore*;*f*) = *fmore*;*f*
using *FiniteChopFmoreEqvFmore 12 13* **by** (*metis int-eq*)
have *2*: $\vdash$ *fmore* ; *f* = $\bigcirc(\Diamond$ *f*)
using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash \Diamond$ (*f*) $\longrightarrow \bigcirc(\Diamond$ *f*)
using *11 14 2* **by** *fastforce*
hence *4*: $\vdash$ *finite* $\longrightarrow \neg$ ($\Diamond$ *f*)
using *NextLoop* **by** *blast*
have *5*: $\vdash \neg$ ($\Diamond$ *f*) $\longrightarrow \neg$ *f*
using *NowImpDiamond* **by** *fastforce*
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *MoreChopContra*:
assumes $\vdash$ *f* $\wedge \neg$ *g* $\longrightarrow$ (*fmore* ; (*f* $\wedge \neg$ *g*))
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *g*
proof −
have *1*: $\vdash$ *f* $\wedge \neg$ *g* $\longrightarrow$ (*fmore* ; (*f* $\wedge \neg$ *g*)) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *finite* $\longrightarrow \neg$ (*f* $\wedge \neg$ *g*) **by** (*rule MoreChopLoop*)
from *2* **show** *?thesis* **by** *auto*
qed

lemma *MoreChopLoopFinite*:
assumes $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *fmore* ; *f*
shows $\vdash$ *finite* $\longrightarrow \neg$ *f*
proof −
have *1*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *fmore* ; *f*
using *assms* **by** *auto*
hence *11*: $\vdash \Diamond$ (*f* $\wedge$ *finite*) $\longrightarrow \Diamond$ (*fmore*;*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash \Diamond$ (*fmore*;*f*) = *finite*;(*fmore*;*f*)
by (*simp add*: *sometimes-d-def*)
have *13*: $\vdash$ *finite*;(*fmore*;*f*) = (*finite*;*fmore*);*f*

by (*rule ChopAssoc*)
have *14*: ⊢ ◇ (*fmore*;*f*) = *fmore*;*f*
using *FiniteChopFmoreEqvFmore 12 13* **by** (*metis int-eq*)
have *2*: ⊢ *fmore* ; *f* = ○(◇ *f*)
using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: ⊢ ◇ (*f* ∧ *finite*) ⟶ ○(◇ *f*)
using *11 14 2* **by** *fastforce*
have *31*: ⊢ ◇ (*f* ∧ *finite*) = ((◇ *f*) ∧ *finite*)
by (*metis* (*no-types*, *lifting*) *3 ChopAndB ChopAndNotChopImp DiamondDiamondEqvDiamond DiamondIntroC FiniteChopFiniteEqvFinite FiniteChopInfEqvInf Prop11 Prop12 finite-d-def inteq-reflection sometimes-d-def*)
have *32*: ⊢ (◇ *f*) ∧ *finite* ⟶ ○(◇ *f*)
using *3 31* **by** *fastforce*
hence *4*: ⊢ *finite* ⟶ ¬ (◇ *f*)
by (*metis* (*no-types*, *lifting*) *DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09 finite-d-def int-simps*(*15*) *int-simps*(*32*) *inteq-reflection sometimes-d-def*)
have *5*: ⊢ ¬ (◇ *f*) ⟶ ¬ *f*
by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreChopEqvFmoreOrInf*:
⊢ *more* ; *f* = ((*fmore*;*f*) ∨ *inf*)
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv OrFiniteInf fmore-d-def int-eq*)

lemma *MoreChopLoopFiniteB*:
assumes ⊢ *f* ⟶ *more* ; *f*
shows ⊢ *finite* ⟶ ¬ *f*
proof −
have *1*: ⊢ *f* ⟶ *more* ; *f*
using *assms* **by** *auto*
have *10*: ⊢ *f* ⟶ (*fmore*;*f*) ∨ *inf*
using *MoreChopEqvFmoreOrInf assms* **by** *fastforce*
hence *100*: ⊢ *f* ∧ *finite* ⟶ (*fmore*;*f*)
by (*simp add*: *Prop13 finite-d-def*)
hence *11*: ⊢ ◇ (*f* ∧ *finite*) ⟶ ◇ (*fmore*;*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: ⊢ ◇ (*fmore*;*f*) = *finite*;(*fmore*;*f*)
by (*simp add*: *sometimes-d-def*)
have *13*: ⊢ *finite*;(*fmore*;*f*) = (*finite*;*fmore*);*f*
by (*rule ChopAssoc*)
have *14*: ⊢ ◇ (*fmore*;*f*) = *fmore*;*f*
using *FiniteChopFmoreEqvFmore 12 13* **by** (*metis int-eq*)
have *2*: ⊢ *fmore* ; *f* = ○(◇ *f*)
using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: ⊢ ◇ (*f* ∧ *finite*) ⟶ ○(◇ *f*)
using *11 14 2* **by** *fastforce*
have *31*: ⊢ ◇ (*f* ∧ *finite*) = ((◇ *f*) ∧ *finite*)
by (*metis* (*no-types*, *opaque-lifting*) *ChopAndA ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFinite*

FiniteChopInfEqvInf Prop11 Prop12 finite-d-def inteq-reflection sometimes-d-def)
have *32*: $\vdash (\Diamond\ f) \wedge$ *finite* $\longrightarrow \bigcirc(\Diamond\ f)$
using *3 31* **by** *fastforce*
hence *4*: $\vdash$ *finite* $\longrightarrow \neg\ (\Diamond\ f)$
by (*metis* (*no-types*, *lifting*) *DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09*
finite-d-def int-simps(*15*) *int-simps*(*32*) *inteq-reflection sometimes-d-def*)
have *5*: $\vdash \neg\ (\Diamond\ f) \longrightarrow \neg\ f$
by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreChopContraFinite*:
assumes $\vdash\ (f \wedge \neg\ g) \wedge$ *finite* $\longrightarrow$ (*fmore* ; $(f \wedge \neg\ g)$)
shows $\vdash f \wedge$ *finite* $\longrightarrow g$
proof −
have *1*: $\vdash (f \wedge \neg\ g) \wedge$ *finite* $\longrightarrow$ (*fmore* ; $(f \wedge \neg\ g)$) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *finite* $\longrightarrow \neg\ (f \wedge \neg\ g)$ **using** *MoreChopLoopFinite* **by** (*simp add*: *MoreChopLoopFinite*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *MoreChopContraFiniteB*:
assumes $\vdash\ (f \wedge \neg\ g)\ \longrightarrow$ (*more* ; $(f \wedge \neg\ g)$)
shows $\vdash f \wedge$ *finite* $\longrightarrow g$
proof −
have *1*: $\vdash (f \wedge \neg\ g)\ \longrightarrow$ (*more* ; $(f \wedge \neg\ g)$) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *finite* $\longrightarrow \neg\ (f \wedge \neg\ g)$ **using** *MoreChopLoopFinite* **by** (*simp add*: *MoreChopLoopFiniteB*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *ChopLoop*:
assumes $\vdash\ f \longrightarrow g;f$
$\vdash\ g \longrightarrow$ *fmore*
shows $\vdash$ *finite* $\longrightarrow \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow g; f$ **using** *assms* **by** *auto*
have *2*: $\vdash\ g \longrightarrow$ *fmore* **using** *assms* **by** *auto*
hence *3*: $\vdash g; f \longrightarrow$ *fmore* ; f **by** (*rule LeftChopImpChop*)
have *4*: $\vdash f \longrightarrow$ *fmore* ; f **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreChopLoop* **by** *auto*
qed

lemma *ChopLoopB*:
assumes $\vdash\ f \longrightarrow g;f$
$\vdash\ g \longrightarrow$ *more*
shows $\vdash$ *finite* $\longrightarrow \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow g; f$ **using** *assms* **by** *auto*
have *2*: $\vdash\ g \longrightarrow$ *more* **using** *assms* **by** *auto*
hence *3*: $\vdash g; f \longrightarrow$ *more* ; f **by** (*rule LeftChopImpChop*)
have *4*: $\vdash f \longrightarrow$ *more* ; f **using** *1 3* **by** *fastforce*

from *4* **show** *?thesis* **using** *MoreChopLoopFiniteB* **by** *auto*
qed

lemma *ChopContra*:
assumes $\vdash \ f \wedge \neg \ g \longrightarrow h;\ f \wedge \neg\ (h;\ g)$
$\vdash h \longrightarrow \ fmore$
shows $\vdash f \ \wedge finite \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg \ g \longrightarrow h;\ f \wedge \neg\ (h;\ g)$ **using** *assms* **by** *auto*
have *2*: $\vdash h \longrightarrow \ fmore$ **using** *assms* **by** *auto*
have *3*: $\vdash h;\ f \wedge \neg\ (h;\ g) \longrightarrow h;\ (f \wedge \neg \ g)$ **by** (*rule ChopAndNotChopImp*)
have *4*: $\vdash h;\ (f \wedge \neg \ g) \longrightarrow \ fmore\ ;\ (f \wedge \neg \ g)$ **using** *2* **by** (*rule LeftChopImpChop*)
have *5*: $\vdash f \wedge \neg \ g \longrightarrow \ fmore\ ;\ (f \wedge \neg \ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreChopContra* **by** *auto*
qed

lemma *ChopContraB*:
assumes $\vdash f \wedge \neg \ g \longrightarrow h;\ f \wedge \neg\ (h;\ g)$
$\vdash h \longrightarrow \ more$
shows $\vdash f \wedge finite \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg \ g \longrightarrow h;\ f \wedge \neg\ (h;\ g)$ **using** *assms* **by** *auto*
have *2*: $\vdash h \longrightarrow \ more$ **using** *assms* **by** *auto*
have *3*: $\vdash h;\ f \wedge \neg\ (h;\ g) \longrightarrow h;\ (f \wedge \neg \ g)$ **by** (*rule ChopAndNotChopImp*)
have *4*: $\vdash h;\ (f \wedge \neg \ g) \longrightarrow \ more\ ;\ (f \wedge \neg \ g)$ **using** *2* **by** (*rule LeftChopImpChop*)
have *5*: $\vdash f \wedge \neg \ g \longrightarrow \ more\ ;\ (f \wedge \neg \ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreChopContraFiniteB* **by** *auto*
qed

3.3.8 Properties of Halt

lemma *WnextAndMoreEqvNext*:
$\vdash (wnext\ f \wedge more) = \bigcirc f$
proof −
have *1*: $\vdash wnext\ f = (empty \vee \bigcirc f)$
by (*simp add*: *WnextEqvEmptyOrNext*)
have *2*: $\vdash \bigcirc f \longrightarrow more$
by (*metis DiIntro LeftChopImpMoreRule di-d-def more-d-def next-d-def*)
have *3*: $\vdash ((empty \vee \bigcirc f) \wedge more) = \bigcirc f$
unfolding *empty-d-def* **using** *2* **by** *auto*
show *?thesis* **by** (*metis 1 3 int-eq*)
qed

lemma *BoxStateAndEmptyEqvStateAndEmpty*:
$\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$
proof −
have *1*: $\vdash ((empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$
by *force*
have *2*: $\vdash (\Box(empty = (init\ w)) \wedge empty) \longrightarrow ((init\ w) \wedge empty)$
using *BoxElim* **by** *fastforce*

have *3*: $\vdash ((init\ w) \wedge empty) \longrightarrow (\Box(empty = (init\ w)) \wedge empty)$
using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*:
$\vdash \Box(empty = (init\ w)) = ((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$
proof −
have *1*: $\vdash \Box(empty = (init\ w)) =$
$((\Box(empty = (init\ w)) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more))$
by (*auto simp*: *empty-d-def*)
have *2*: $\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$
using *BoxStateAndEmptyEqvStateAndEmpty* **by** *blast*
have *3*: $\vdash \Box(empty = (init\ w)) = ((empty = (init\ w)) \wedge wnext(\Box(empty = (init\ w))))$
using *BoxEqvAndWnextBox* **by** *blast*
hence *4*: $\vdash (\Box(empty = (init\ w)) \wedge more) =$
$(((empty = (init\ w)) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$
by *auto*
have *5*: $\vdash ((empty = (init\ w)) \wedge more) = (\neg(init\ w) \wedge more)$
by (*auto simp*: *empty-d-def*)
have *6*: $\vdash (wnext(\Box(empty = (init\ w))) \wedge more) = \bigcirc(\Box(empty = (init\ w)))$
using *WnextAndMoreEqvNext* **by** *metis*
have *7*: $\vdash (\Box(empty = (init\ w)) \wedge more) =$
$((\neg\ (init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ )$
using *4 5* **by** *fastforce*
have *8*: $\vdash ((\neg\ (init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ ) =$
$((\neg\ (init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ )$
by *auto*
have *9*: $\vdash ((\neg\ (init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ ) =$
$((\neg\ (init\ w)) \wedge \bigcirc(\Box(empty = (init\ w))))$
using *8 6* **by** *auto*
have *10*: $\vdash \Box(empty = (init\ w)) = (((init\ w) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more)\ )$
using *1 2* **by** *fastforce*
show *?thesis*
using *10 7 9* **by** *fastforce*
qed

lemma *HaltStateEqvIfStateThenEmptyElseNext*:
$\vdash\ halt(\ init\ w) =\ if_i\ \ (init\ w)\ \ then\ \ \ empty\ \ \ else\ (\ \bigcirc(\ halt\ (\ init\ w)))$
by (*metis BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext halt-d-def ifthenelse-d-def inteq-reflection lift-and-com*)

lemma *HaltChopEqv*:
$\vdash\ ((halt\ (\ init\ w))\ ;\ f) =\ \ (if_i\ \ \ (init\ w)\ \ \ then\ (\ f\ )\ else\ (\bigcirc(\ (halt\ (\ init\ w));\ f)))$
proof −
have *1*: $\vdash\ \ halt(init\ w) =$
$(if_i\ \ \ (init\ w)\ \ \ then\ \ \ \ empty\ \ \ \ else\ (\ \bigcirc(\ \ halt\ (\ init\ w))))$
by (*rule HaltStateEqvIfStateThenEmptyElseNext*)
hence *2*: $\vdash\ \ ((halt(init\ w));f) =$

$(if_i\ \ (init\ w)\ \ then\ \ (empty;f)\ \ \ else\ (\ \bigcirc(\ halt\ (\ init\ w));f))$
by (*rule IfChopEqvRule*)
have *3*: $\vdash\ \ empty\ ;\ f = f$
by (*rule EmptyChop*)
have *4*: $\vdash (\bigcirc\ (halt\ (\ init\ w)));\ f = \bigcirc(\ halt\ (\ init\ w);\ f)$
by (*rule NextChop*)
from *2 3 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *AndHaltChopImp*:
$\vdash\ \ init\ w \wedge (\ halt\ (\ init\ w);\ f) \longrightarrow f$
proof −
have *1*: $\vdash halt\ (\ init\ w);\ f =\ \ if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f))$
by (*rule HaltChopEqv*)
have *2*: $\vdash init\ w \wedge if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f)) \longrightarrow f$
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltChopImpNext*:
$\vdash\ \neg\ (\ init\ w) \wedge (\ halt\ \ (init\ w);\ f) \longrightarrow \bigcirc(\ halt\ (\ init\ w);\ f)$
proof −
have *1*: $\vdash\ \ halt\ (\ init\ w);\ f =\ \ if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f))$
by (*rule HaltChopEqv*)
have *2*: $\vdash \neg\ (\ init\ w) \wedge if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f)) \longrightarrow$
$\bigcirc(\ halt\ (\ init\ w);\ f)$
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltChopImpSkipYields*:
$\vdash\ \neg\ (\ init\ w) \wedge (\ halt\ (\ init\ w);\ f) \longrightarrow\ \ skip\ \ yields\ (\ halt\ \ (init\ w);\ f)$
proof −
have *1*: $\vdash \neg\ (\ init\ w) \wedge (\ halt\ (\ init\ w);\ f) \longrightarrow \bigcirc(\ halt\ (\ init\ w);\ f)$
by (*rule NotAndHaltChopImpNext*)
have *2*: $\vdash \bigcirc(\ halt\ (\ init\ w);\ f) \longrightarrow\ \ skip\ \ yields\ (\ halt\ (\ init\ w);\ f)$
by (*rule NextImpSkipYields*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteChopAndEmptyEqvChopAndEmpty*:
$\vdash ((finite;(f \wedge empty)) \wedge g) = ((g \wedge finite);(f \wedge empty))$
proof −
have *1*: $\vdash g \wedge finite;(f \wedge empty) \longrightarrow fin\ f$
by (*metis ChopAndA DiamondFin FinAndEmpty Prop01 Prop05 inteq-reflection sometimes-d-def*)
have *2*: $\vdash g \wedge finite;(f \wedge empty) \longrightarrow (finite \wedge g) \wedge fin\ f$
using *1* **by** (*metis* (*no-types*, *lifting*) *ChopAndB ChopEmpty Prop10 Prop12 int-iffD1*
inteq-reflection)
have *3*: $\vdash ((finite;(f \wedge empty)) \wedge g) \longrightarrow ((g \wedge finite);(f \wedge empty))$
using *2* **using** *AndFinEqvChopAndEmpty* **by** *fastforce*

have *4*: $\vdash ((g \wedge \mathit{finite});(f \wedge \mathit{empty})) \longrightarrow ((\mathit{finite};(f \wedge \mathit{empty})) \wedge g)$
by (*metis AndChopB ChopAndB ChopEmpty Prop12 inteq-reflection*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *WprevEqvEmptyOrPrev*:
$\vdash \mathit{wprev}\ f = (\mathit{empty} \vee \mathit{prev}\ f)$
using *nlength-eq-enat-nfiniteD* **by** (*auto simp add*: *Valid-def itl-defs zero-enat-def*)

lemma *NotChopSkipEqvMoreAndNotChopSkip*:
$\vdash (\neg f);\mathit{skip} = (\mathit{more} \wedge \neg(f;\mathit{skip}))$
proof −
have *1*: $\vdash \mathit{wprev}\ f = (\mathit{empty} \vee \mathit{prev}\ f)$ **using** *WprevEqvEmptyOrPrev* **by** *auto*
hence *2*: $\vdash (\neg(\mathit{wprev}\ f)) = (\neg(\mathit{empty} \vee \mathit{prev}\ f))$ **by** *auto*
have *3*: $\vdash \neg(\mathit{wprev}\ f) = ((\neg f);\mathit{skip})$ **by** (*simp add*: *wprev-d-def prev-d-def*)
have *31*: $\vdash (\mathit{empty} \vee \mathit{prev}\ f) = (\mathit{empty} \vee (f;\mathit{skip}))$ **by** (*simp add*: *prev-d-def*)
have *32*: $\vdash (\mathit{empty} \vee (f;\mathit{skip})) = (\neg \mathit{more} \vee \neg\neg(f;\mathit{skip}))$ **by** (*simp add*: *empty-d-def*)
have *33*: $\vdash (\neg \mathit{more} \vee \neg\neg(f;\mathit{skip})) = (\neg(\mathit{more} \wedge \neg(f;\mathit{skip})))$ **by** *fastforce*
have *34*: $\vdash (\mathit{empty} \vee \mathit{prev}\ f) = (\neg(\mathit{more} \wedge \neg(f;\mathit{skip})))$ **using** *31 32 33* **by** (*metis int-eq*)
have *4*: $\vdash \neg(\mathit{empty} \vee \mathit{prev}\ f) = (\mathit{more} \wedge \neg(f;\mathit{skip}))$ **using** *34* **by** *fastforce*
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltChopImpNotHaltChopNot*:
$\vdash \quad \mathit{halt}\ (\ \mathit{init}\ w);\ f \wedge \mathit{finite} \longrightarrow \neg\ (\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg\ \ f))$
proof −
have *1*: $\vdash \ \mathit{halt}\ \ (\mathit{init}\ w);\ f = \ \ \mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ f\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f))$
by (*rule HaltChopEqv*)
have *2*: $\vdash \mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ f\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \longrightarrow$
$(\ ((\mathit{init}\ w) \longrightarrow f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f))))$
by (*rule IfThenElseImp*)
have *3*: $\vdash \ \mathit{halt}\ \ (\mathit{init}\ w);\ (\neg f) =$
$\mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ (\neg f)\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))$
by (*rule HaltChopEqv*)
have *4*: $\vdash \mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ (\neg f)\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f))) \longrightarrow$
$(\ ((\mathit{init}\ w) \longrightarrow \neg f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))))$
by (*rule IfThenElseImp*)
have *5*: $\vdash \mathit{halt}\ \ (\mathit{init}\ w);\ f \wedge \mathit{halt}\ \ (\mathit{init}\ w);\ (\neg f) \longrightarrow$
$(\ ((\mathit{init}\ w) \longrightarrow f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)))) \wedge$
$(\ ((\mathit{init}\ w) \longrightarrow \neg f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))))$
using *1 2 3 4* **by** *fastforce*
have *6*: $\vdash (\ ((\mathit{init}\ w) \longrightarrow f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)))) \wedge$
$(\ ((\mathit{init}\ w) \longrightarrow \neg f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f))))) \longrightarrow$
$(\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \wedge (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))$
by *auto*
have *7*: $\vdash \mathit{halt}\ \ (\mathit{init}\ w);\ f \wedge \mathit{halt}\ \ (\mathit{init}\ w);\ (\neg f) \longrightarrow$
$(\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \wedge (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))$
using *5 6 lift-imp-trans* **by** *blast*
have *8*: $\vdash ((\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \wedge (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))) =$
$\bigcirc\ (\mathit{halt}\ (\ \mathit{init}\ w);\ f \wedge \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f))$

using *NextAndEqvNextAndNext* **by** *fastforce*
have *9*: $\vdash$ *halt* (*init w*); $f \wedge$ *halt* (*init w*); $(\neg f) \longrightarrow$
$\bigcirc$ (*halt* (*init w*); $f \wedge$ *halt* (*init w*); $(\neg f)$)
using *7 8* **by** *fastforce*
hence *10*: $\vdash$ *finite* $\longrightarrow \neg$(*halt* (*init w*); $f \wedge$ *halt* (*init w*); $(\neg f)$)
using *NextLoop* **by** *blast*
from *10* **show** *?thesis* **by** *auto*
qed

lemma *HaltChopImpHaltYields*:
$\vdash$ *halt* (*init w*); $f \wedge$ *finite* $\longrightarrow$ (*halt* (*init w*)) *yields* f
proof −
have *1*: $\vdash$ *halt* (*init w*); $f \wedge$ *finite* $\longrightarrow \neg$ (*halt* (*init w*); $(\neg\ f)$)
by (*rule HaltChopImpNotHaltChopNot*)
from *1* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *HaltChopAnd*:
$\vdash$ (*halt* (*init w*)); $f \wedge$ (*halt* (*init w*)); $g \wedge$ *finite*$\longrightarrow$ (*halt* (*init w*)); $(f \wedge g)$
proof −
have *1*: $\vdash$ (*halt* (*init w*)); $g \wedge$ *finite* $\longrightarrow$ (*halt* (*init w*)) *yields* g
by (*rule HaltChopImpHaltYields*)
hence *2*: $\vdash$ (*halt* (*init w*)); $f \wedge$ (*halt* (*init w*)); $g \wedge$ *finite* $\longrightarrow$
(*halt* (*init w*)); $f \wedge$ (*halt* (*init w*)) *yields* g
by *auto*
have *3*: $\vdash$ (*halt* (*init w*)); $f \wedge$ (*halt* (*init w*)) *yields* $g\longrightarrow$
(*halt* (*init w*)); $(f \wedge g)$
by (*rule ChopAndYieldsImp*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltAndChopAndHaltChopImpHaltAndChopAnd*:
$\vdash$ (*halt* (*init w*) $\wedge$ *f*);*f1* $\wedge$ (*halt* (*init w*); *g*) $\wedge$ *finite* $\longrightarrow$ (*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *f1* $\longrightarrow \neg\ g \vee$ (*f1* $\wedge$ *g*)
by *auto*
hence *2*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); *f1* $\longrightarrow$
(*halt* (*init w*) $\wedge$ *f*); $(\neg\ g) \vee$ ((*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*))
by (*rule ChopOrImpRule*)
have *3*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); $(\neg\ g) \longrightarrow$ *halt* (*init w*); $(\neg\ g)$
by (*rule AndChopA*)
have *31*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); *f1* $\longrightarrow$
halt (*init w*); $(\neg\ g) \vee$ ((*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*))
using *23* **by** *fastforce*
have *4*: $\vdash$ *halt* (*init w*); $g \wedge$ *finite* $\longrightarrow \neg$ (*halt* (*init w*); $(\neg\ g)$)
by (*rule HaltChopImpNotHaltChopNot*)
hence *41*: $\vdash$ (*halt* (*init w*); $(\neg\ g)) \wedge$ *finite* $\longrightarrow \neg$(*halt* (*init w*); *g*)
by *auto*
have *42*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); *f1* $\wedge$ *finite* $\longrightarrow$
$\neg$(*halt* (*init w*); *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*))

using *31 41* **by** *fastforce*
from *42* **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxYields*:
$\vdash$ (*halt* (*init w*)); *f* $\wedge$ *finite* $\longrightarrow$ (□(¬ (*init w*))) *yields* ((*halt* (*init w*)); *f*)
proof −
have *1*: $\vdash$ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) $\longrightarrow$ *di* (□ (¬ (*init w*)))
by (*rule ChopImpDi*)
have *2*: $\vdash$ □ (¬ (*init w*)) $\longrightarrow$ ¬ (*init w*)
by (*rule BoxElim*)
hence *3*: $\vdash$ *di* (□ (¬ (*init w*))) $\longrightarrow$ *di* (¬ (*init w*))
by (*rule DiImpDi*)
have *4*: $\vdash$ *di* (*init* (¬ *w*)) = (*init* (¬*w*))
by (*rule DiState*)
have *41*: $\vdash$ (*init* (¬ *w*)) = (¬ (*init w*))
using *Initprop*(*2*) **by** *fastforce*
have *42*: $\vdash$ *di* (¬ (*init w*)) = (¬(*init w*))
using *4 41* **by** (*metis inteq-reflection*)
have *5*: $\vdash$ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) $\longrightarrow$ ¬ (*init w*)
using *1 2 42* **using** *3* **by** *fastforce*
hence *51*: $\vdash$ (*halt* (*init w*); *f*) $\wedge$ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) $\longrightarrow$
(*halt* (*init w*); *f*) $\wedge$ ¬ (*init w*)
by *fastforce*
have *6*: $\vdash$ *halt* (*init w*); *f* = *if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*))
by (*rule HaltChopEqv*)
hence *61*: $\vdash$ (*halt* (*init w*); *f* $\wedge$ ¬ (*init w*)) =
((*if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*))) $\wedge$ ¬ (*init w*))
using *6* **by** *auto*
have *62*: $\vdash$ (*if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*))) $\wedge$
¬ (*init w*) $\longrightarrow$ (○(*halt* (*init w*); *f*))
by (*auto simp*: *ifthenelse-d-def*)
have *63*: $\vdash$ *halt* (*init w*); *f* $\wedge$ ¬ (*init w*) $\longrightarrow$ (○(*halt* (*init w*); *f*))
using *61 62* **by** *fastforce*
have *7*: $\vdash$ (*halt* (*init w*); *f*) $\wedge$ (□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) $\longrightarrow$
○((*halt* (*init w*)); *f*)
using *51 63* **using** *lift-imp-trans* **by** *blast*
have *8*: $\vdash$ □ (¬ (*init w*)) $\longrightarrow$ *empty* $\vee$ ○(□(¬(*init w*)))
by (*metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq*)
hence *9*: $\vdash$ ((□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) $\longrightarrow$
¬ (*halt* (*init w*); *f*) $\vee$ ○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
by (*rule EmptyOrNextChopImpRule*)
hence *10*: $\vdash$ ((*halt* (*init w*)); *f*) $\wedge$ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) $\longrightarrow$
○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
by *fastforce*
have *11*: $\vdash$ (*halt* (*init w*)); *f* $\wedge$ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) $\longrightarrow$
○((*halt* (*init w*)); *f*) $\wedge$ ○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
using *7 10* **by** *fastforce*
have *12*: $\vdash$ ○((*halt* (*init w*)); *f*) $\wedge$ ○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
$\longrightarrow$ ○(((*halt* (*init w*)); *f*) $\wedge$ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))))

using *NextAndEqvNextAndNext* **by** *fastforce*
have *13*: $\vdash$ (*halt* (*init w*)); *f* $\wedge$ ($\Box$ ($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*)) $\longrightarrow$
$\bigcirc$(((*halt* (*init w*)); *f*) $\wedge$ (($\Box$($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*))))
using *11 12* **by** *fastforce*
hence *14*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ((*halt* (*init w*)); *f* $\wedge$ ($\Box$ ($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*)))
using *NextLoop* **by** *blast*
hence *15*: $\vdash$ (*halt* (*init w*)); *f* $\wedge$ *finite* $\longrightarrow$ $\neg$ (($\Box$ ($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*)))
by *auto*
from *15* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

3.3.9 Properties of Groups of chops

lemma *NestedChopImpChop*:
assumes $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*; (*init w1* $\wedge$ *f1*)
$\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*; (*init w2* $\wedge$ *f2*)
shows $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*; (*g1*; (*init w2* $\wedge$ *f2*))
proof −
have *1*: $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*; (*init w1* $\wedge$ *f1*) **using** *assms*(*1*) **by** *auto*
have *2*: $\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*; (*init w2* $\wedge$ *f2*) **using** *assms*(*2*) **by** *auto*
hence *3*: $\vdash$ *g*; (*init w1* $\wedge$ *f1*) $\longrightarrow$ *g*; (*g1*; (*init w2* $\wedge$ *f2*)) **by** (*rule RightChopImpChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

end

3.4 Strong chop operator

theory *SChopTheorems*
imports
Theorems
begin

We give the proofs of a list of Finite and Infinite ITL theorems but now using the strong chop.

3.4.1 Strong Chop axioms

lemma *SChopAssoc*:
$\vdash$ *f*⌢(*g*⌢*h*) = (*f*⌢*g*)⌢*h*
proof −
have *1*: $\vdash$ *f*⌢(*g*⌢*h*) = (*f* $\wedge$ *finite*);((*g* $\wedge$ *finite*);*h*)
by (*simp add*: *schop-d-def*)
have *2*: $\vdash$ (*f* $\wedge$ *finite*);((*g* $\wedge$ *finite*);*h*) = ((*f* $\wedge$ *finite*);(*g* $\wedge$ *finite*));*h*
using *ChopAssoc* **by** *blast*
have *3*: $\vdash$ ((*f* $\wedge$ *finite*);(*g* $\wedge$ *finite*));*h* = (*f*⌢ (*g* $\wedge$ *finite*));*h*
by (*simp add*: *schop-d-def*)
have *4*: $\vdash$ *f*⌢ (*g* $\wedge$ *finite*) = (*f*⌢*g* $\wedge$ *finite*)
by (*simp add*: *schop-d-def*)
(*metis AndChopA ChopAndA ChopAndFiniteDist Prop11 Prop12 inteq-reflection*)

have *5*: $\vdash (f \frown (g \wedge \mathit{finite}));h = (f \frown g \wedge \mathit{finite});h$
using *4* **by** (*simp add*: *LeftChopEqvChop*)
have *6*: $\vdash (f \frown g \wedge \mathit{finite});h = (f \frown g) \frown h$
by (*simp add*: *schop-d-def*)
from *1 2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteOr*:
$\vdash ((f \vee g) \wedge \mathit{finite}) = ((f \wedge \mathit{finite}) \vee (g \wedge \mathit{finite}))$
by *auto*

lemma *OrSChopImp* :
$\vdash (f \vee g) \frown h \longrightarrow f \frown h \vee g \frown h$
unfolding *schop-d-def*
by (*simp add*: *FiniteOr OrChopImpRule int-iffD1*)

lemma *SChopOrImp* :
$\vdash f \frown (g \vee h) \longrightarrow f \frown g \vee f \frown h$
unfolding *schop-d-def* **by** (*simp add*: *ChopOrImp*)

lemma *EmptySChop* :
$\vdash \mathit{empty} \frown f = f$
by (*metis EmptyChopSem FiniteAndEmptyEqvEmpty intI inteq-reflection lift-and-com schop-d-def*)

lemma *SChopEmpty* :
$\vdash \mathit{finite} \longrightarrow f \frown \mathit{empty} = f$
unfolding *schop-d-def*
proof −
have *f1*: $\vdash (f \wedge \mathit{finite});\mathit{empty} = (f \wedge \mathit{finite})$
by (*simp add*: *ChopEmpty int-eq*)
then show $\vdash \mathit{finite} \longrightarrow (f \wedge \mathit{finite});\mathit{empty} = f$
by *fastforce*
qed

lemma *StateImpBf* :
$\vdash \mathit{init}\ f \longrightarrow \mathit{bf}\ (\mathit{init}\ f)$
unfolding *bf-d-def df-d-def schop-d-def*
by (*metis* (*no-types*) *AndChopA StateImpBi bi-d-def di-d-def lift-imp-neg lift-imp-trans*)

lemma *BfBoxSChopImpSChop* :
$\vdash \mathit{bf}\ (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f1 \frown g1$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndMoreSChopEqvAndFmoreChop*:
$\vdash (f \wedge \mathit{more}) \frown g = (f \wedge \mathit{fmore});g$
by (*simp add*: *LeftChopEqvChop AndMoreAndFiniteEqvAndFmore schop-d-def*)

lemma *FiniteBfGen*:
assumes $\vdash \mathit{finite} \longrightarrow f$

shows $\vdash bf\ f$
using *assms*
by (*simp add*: *Valid-def itl-defs*)

lemma *BfGen*:
assumes $\vdash f$
shows $\vdash bf\ f$
using *assms*
by (*metis EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteBfGen Prop09 int-eq-true inteq-reflection*)

3.4.2 ITL operators in terms of SChop

lemma *NextSChopdef*:
$\vdash \bigcirc f = skip^\frown f$
by (*metis FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 SkipChopFiniteImpFinite inteq-reflection lift-imp-trans next-d-def schop-d-def sometimes-d-def*)

lemma *DiamondSChopdef*:
$\vdash \Diamond f = \#True^\frown f$
by (*simp add*: *schop-d-def sometimes-d-def*)

lemma *FiniteSChopdef*:
$\vdash finite = \Diamond\ empty$
by (*simp add*: *DiamondEmptyEqvFinite int-iffD1 int-iffD2 int-iffI*)

lemma *ChopSChopdef*:
$\vdash f;g = ((f^\frown g) \vee (f \wedge inf))$
by (*metis AndInfChopEqvAndInf OrChopEqv OrFiniteInf inteq-reflection schop-d-def*)

lemma *SFinprop* :
$\vdash ((\#True^\frown(f \wedge empty)) \wedge (\#True^\frown(g \wedge empty))) = (\#True^\frown((f \wedge g) \wedge empty))$
$\vdash ((\#True^\frown(f \wedge empty)) \vee (\#True^\frown(g \wedge empty))) = (\#True^\frown((f \vee g) \wedge empty))$
$\vdash\ finite \longrightarrow (\neg\ (\#True^\frown(f \wedge empty))) = (\#True^\frown(\neg f \wedge empty))$
$\vdash (\neg\ (\#True^\frown(f \wedge empty))) = ((\#True^\frown(\neg f \wedge empty)) \vee inf)$
by (*auto simp add*: *Valid-def itl-defs zero-enat-def*)
(*metis add.right-neutral enat.distinct(2) enat-add-sub-same less-eqE the-enat.simps zero-enat-def*,
metis ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat nlength-NNil the-enat.simps zero-enat-def,
metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add the-enat.simps zero-enat-def,
metis ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat nlength-NNil the-enat.simps zero-enat-def,
metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add the-enat.simps zero-enat-def,
metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD)

3.4.3 Basic Theorems

lemma *BfSChopImpSChop* :
$\vdash bf\ (f \longrightarrow f1) \longrightarrow f^\frown g \longrightarrow f1^\frown g$

proof −
have *1*: $\vdash g \longrightarrow g$ **by** *auto*
hence *2*: $\vdash \Box\ (\ g \longrightarrow g)$ **by** (*rule BoxGen*)
have *3*: $\vdash bf\ (\ f \longrightarrow f1) \wedge \Box(g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfBoxSChopImpSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiImpBf*:
$\vdash bi\ f \longrightarrow bf\ f$
unfolding *bi-d-def bf-d-def di-d-def df-d-def schop-d-def*
by (*simp add*: *AndChopA*)

lemma *BiSChopImpSChop* :
$\vdash bi\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof −
have *1*: $\vdash g \longrightarrow g$
by *auto*
hence *2*: $\vdash \Box\ (\ g \longrightarrow g)$
by (*rule BoxGen*)
have *3*: $\vdash bi\ (\ f \longrightarrow f1) \wedge \Box(g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$
using *BiImpBf BfBoxSChopImpSChop* **using** *BfSChopImpSChop* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *AndSChopA*:
$\vdash (f \wedge f1) \frown g \longrightarrow f \frown g$
proof −
have *1*: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*
hence *2*: $\vdash bf\ (f \wedge f1 \longrightarrow f)$ **by** (*rule BfGen*)
have *3*: $\vdash bf\ (f \wedge f1 \longrightarrow f) \longrightarrow (f\ \wedge f1) \frown g \longrightarrow f \frown g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *AndSChopB*:
$\vdash (f \wedge f1) \frown g \longrightarrow f1 \frown g$
proof −
have *1*: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*
hence *2*: $\vdash\ bf\ (f \wedge f1 \longrightarrow f1)$ **by** (*rule BfGen*)
have *3*: $\vdash\ bf\ (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1) \frown g \longrightarrow f1 \frown g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *NextSChop*:
$\vdash (\bigcirc\ f) \frown g\ = \bigcirc(f \frown g)$
proof −
have *1*: $\vdash skip \frown (f \frown g)\ = (skip \frown f) \frown g$ **by** (*rule SChopAssoc*)
from *1* **show** *?thesis* **using** *NextSChopdef* **by** (*metis inteq-reflection*)

qed

lemma *BoxSChopImpSChop* :
$\vdash \Box\ (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$
proof −
have *1*: $\vdash g \longrightarrow g$ **by** *auto*
hence *2*: $\vdash bf\ (\ g \longrightarrow g)$ **by** (*rule BfGen*)
have *3*: $\vdash bf\ (\ f \longrightarrow f) \wedge \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BfBoxSChopImpSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftSChopImpSChop*:
assumes $\vdash f \longrightarrow f1$
shows $\vdash f \frown g \longrightarrow f1 \frown g$
proof −
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash bf\ (f \longrightarrow f1)$ **by** (*rule BfGen*)
have *3*: $\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightSChopImpSChop*:
assumes $\vdash g \longrightarrow g1$
shows $\vdash f \frown g \longrightarrow f \frown g1$
proof −
have *1*: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ (g \longrightarrow g1)$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box\ (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BoxSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightSChopEqvSChop*:
assumes $\vdash g = g1$
shows $\vdash (f \frown g) = (f \frown g1)$
proof −
have *1*: $\vdash g = g1$ **using** *assms* **by** *auto*
have *2*: $(\vdash g \longrightarrow g1) \Longrightarrow (\vdash f \frown g \longrightarrow f \frown g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $(\vdash g1 \longrightarrow g) \Longrightarrow (\vdash f \frown g1 \longrightarrow f \frown g)$ **by** (*rule RightSChopImpSChop*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxRightSChopEqvSChop*:
$\vdash \Box\ (g = g1) \longrightarrow (f \frown g) = (f \frown g1)$
proof −
have *0*: $\vdash (g = g1) = (\ (g \longrightarrow g1) \wedge\ \ (\ g1 \longrightarrow g))$

by *fastforce*
have *1*: $\vdash \Box\ (g = g1) = (\Box\ (g \longrightarrow g1) \wedge \Box\ (\ g1 \longrightarrow g))$
by (*metis 0 BoxAndBoxEqvBoxRule inteq-reflection*)
have *2*: $\vdash \Box\ (g \longrightarrow g1) \longrightarrow (f \frown g) \longrightarrow (f \frown g1)$
by (*simp add*: *BoxSChopImpSChop*)
have *3*: $\vdash \Box\ (g1 \longrightarrow g) \longrightarrow (f \frown g1) \longrightarrow (f \frown g)$
by (*simp add*: *BoxSChopImpSChop*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteRightSChopEqvSChop*:
assumes $\vdash$ *finite* $\longrightarrow g = g1$
shows $\vdash$ *finite* $\longrightarrow (f \frown g) = (f \frown g1)$
using *assms* **unfolding** *schop-d-def*
by (*simp add*: *FiniteRightChopEqvChop*)

lemma *SChopOrEqv*:
$\vdash f \frown (g \vee g1) = (f \frown g \vee f \frown g1)$
proof −
have *1*: $\vdash g \longrightarrow g \vee g1$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown (g \vee g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash g1 \longrightarrow g \vee g1$ **by** *auto*
hence *4*: $\vdash f \frown g1 \longrightarrow f \frown (g \vee g1)$ **by** (*rule RightSChopImpSChop*)
from *2 4* **show** *?thesis* **by** (*meson SChopOrImp Prop02 Prop11*)
qed

lemma *OrSChopEqv*:
$\vdash (f \vee f1) \frown g = (f \frown g \vee f1 \frown g)$
proof −
have *1*: $\vdash f \longrightarrow f \vee f1$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (f \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*
hence *4*: $\vdash f1 \frown g \longrightarrow (f \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
from *2 4* **show** *?thesis*
by (*meson OrSChopImp int-iffI Prop02*)
qed

lemma *OrSChopImpRule*:
assumes $\vdash f \longrightarrow f1 \vee f2$
shows $\vdash f \frown g \longrightarrow (f1 \frown g) \vee (f2 \frown g)$
proof −
have *1*: $\vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (f1 \vee f2) \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash (f1 \vee f2) \frown\ g = (f1 \frown g \vee f2 \frown g)$ **by** (*rule OrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftSChopEqvSChop*:

assumes $\vdash f = f1$
shows $\vdash f \frown g = (f1 \frown g)$
proof −
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow f1$ **by** *auto*
hence *3*: $\vdash f \frown g \longrightarrow f1 \frown g$ **by** (*rule LeftSChopImpSChop*)
have $\vdash f1 \longrightarrow f$ **using** *1* **by** *auto*
hence *4*: $\vdash f1 \frown g \longrightarrow f \frown g$ **by** (*rule LeftSChopImpSChop*)
from *3 4* **show** *?thesis* **by** (*simp add*: *int-iffI*)
qed

lemma *OrSChopEqvRule*:
assumes $\vdash f = (f1 \vee f2)$
shows $\vdash f \frown g = ((f1 \frown g) \vee (f2 \frown g))$
proof −
have *1*: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = ((f1 \vee f2) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (f1 \vee f2) \frown g = (f1 \frown g \vee f2 \frown g)$ **by** (*rule OrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopOrImpRule*:
assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f \frown g \longrightarrow (f \frown g1) \vee (f \frown g2)$
proof −
have *1*: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown (g1 \vee g2)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown (g1 \vee g2) = (f \frown g1 \vee f \frown g2)$ **by** (*rule SChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopImpDiamond*:
$\vdash f \frown g \longrightarrow \Diamond\, g$
proof −
have *1*: $\vdash f \longrightarrow \#True$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow \#True \frown g$ **by** (*rule LeftSChopImpSChop*)
from *2* **show** *?thesis* **using** *DiamondSChopdef* **by** *fastforce*
qed

lemma *BfImpDfImpDf*:
$\vdash bf\ (f \longrightarrow g) \longrightarrow\ df\ f \longrightarrow\ df\ g$
proof −
have *1*: $\vdash bf\ (f \longrightarrow g) \longrightarrow (f \frown \#True) \longrightarrow (g \frown \#True)$ **by** (*rule BfSChopImpSChop*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfImpDf*:

assumes $\vdash f \longrightarrow g$
shows $\vdash$ *df f* $\longrightarrow$ *df g*
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown \#True \longrightarrow g \frown \#True$ **by** (*rule LeftSChopImpSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *BfImpBfRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash$ *bf f* $\longrightarrow$ *bf g*
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow \neg\ f$ **by** *auto*
hence *3*: $\vdash$ *df* $(\neg\ g) \longrightarrow$ *df* $(\neg\ f)$ **by** (*rule DfImpDf*)
hence *4*: $\vdash \neg\ (\ df\ (\neg\ f)) \longrightarrow \neg\ (\ df\ (\neg\ g))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

lemma *DfEqvDf*:
assumes $\vdash f = g$
shows $\vdash$ *df f* = *df g*
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown \#True = g \frown \#True$ **by** (*rule LeftSChopEqvSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *BfEqvBf*:
assumes $\vdash f = g$
shows $\vdash$ *bf f* = *bf g*
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ f) = (\neg\ g)$ **by** *auto*
hence *3*: $\vdash$ *df* $(\neg\ f) =$ *df* $(\neg\ g)$ **by** (*rule DfEqvDf*)
hence *4*: $\vdash (\neg\ (df\ (\neg\ f))) = (\neg\ (\ df\ (\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

lemma *LeftSChopSChopImpSChopRule*:
assumes $\vdash (f \frown g) \longrightarrow g$
shows $\vdash (f \frown g) \frown h \longrightarrow (g \frown h)$
proof −
have *1*: $\vdash (f \frown g) \longrightarrow g$ **using** *assms* **by** *blast*
hence *2*: $\vdash (f \frown g) \frown h \longrightarrow g \frown h$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash f \frown (g \frown h) = (f \frown g) \frown h$ **by** (*rule SChopAssoc*)
from *2 3* **show** *?thesis* **by** *auto*

qed

lemma *AndSChopCommute* :
$\vdash (f \wedge f1)^\frown\ g = (f1 \wedge f)^\frown\ g$
proof −
have *1*: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftSChopEqvSChop*)
qed

lemma *BfAndSChopImport*:
$\vdash$ *bf f* $\wedge$ $(f1^\frown\ g) \longrightarrow (f \wedge f1)^\frown g$
proof −
have *1*: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence *2*: $\vdash$ *bf f* $\longrightarrow$ *bf* $(f1 \longrightarrow f \wedge f1)$ **by** (*rule BfImpBfRule*)
have *3*: $\vdash$ *bf* $(f1 \longrightarrow (f \wedge f1)) \longrightarrow f1^\frown\ g \longrightarrow (f \wedge f1)^\frown\ g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *BiAndSChopImport*:
$\vdash$ *bi f* $\wedge$ $(f1^\frown\ g) \longrightarrow (f \wedge f1)^\frown g$
proof −
have *1*: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence *2*: $\vdash$ *bi f* $\longrightarrow$ *bi* $(f1 \longrightarrow f \wedge f1)$ **by** (*rule BiImpBiRule*)
have *3*: $\vdash$ *bi* $(f1 \longrightarrow (f \wedge f1)) \longrightarrow f1^\frown\ g \longrightarrow (f \wedge f1)^\frown\ g$ **by** (*rule BiSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *StateAndSChopImport*:
$\vdash$ $(init\ w) \wedge (f^\frown g) \longrightarrow ((init\ w) \wedge f)^\frown g$
proof −
have *1*: $\vdash$ $(init\ w) \longrightarrow$ *bf* $(init\ w)$ **by** (*rule StateImpBf*)
hence *2*: $\vdash$ $(init\ w) \wedge (f^\frown\ g) \longrightarrow$ *bf* $(init\ w) \wedge (f^\frown\ g)$ **by** *auto*
have *3*: $\vdash$ *bf* $(init\ w) \wedge (f^\frown\ g) \longrightarrow ((init\ w) \wedge f)^\frown\ g$ **by** (*rule BfAndSChopImport*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

3.4.4 Further Properties Df and Bf

lemma *AndFiniteImpDf*:
$\vdash f \wedge$ *finite* $\longrightarrow$ *df f*
proof −
have *1*: $\vdash$ *finite* $\longrightarrow f^\frown$ *empty* $= f$ **by** (*rule SChopEmpty*)
have *2*: $\vdash$ *empty* $\longrightarrow \#True$ **by** *auto*
hence *3*: $\vdash f^\frown$ *empty* $\longrightarrow f^\frown \#True$ **by** (*rule RightSChopImpSChop*)
have *4* : $\vdash f \wedge$ *finite* $\longrightarrow f^\frown \#True$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfState*:
$\vdash$ *df* (*init w*) = (*init w*)
proof −
have *0*: $\vdash$ (*init* (¬*w*)) ⟶ *bf* (*init* (¬*w*)) **using** *StateImpBf* **by** *fastforce*
hence *1*: $\vdash$ ¬(*init w*) ⟶ *bf* (¬ (*init w*)) **using** *Initprop*(*2*) **by** (*metis inteq-reflection*)
hence *2*: $\vdash$ (¬ (*init w*)) ⟶ ¬ (*df* (¬ ¬ (*init w*))) **by** (*simp add*: *bf-d-def*)
have *3*: $\vdash$ (¬ (*init w*) ⟶¬ (*df* (¬¬(*init w*)))) ⟶ (*df* (¬¬(*init w*)) ⟶ (*init w*)) **by** *auto*
have *4*: $\vdash$ *df* (¬ ¬ (*init w*)) ⟶ (*init w*) **using** *2 3 MP* **by** *blast*
have *5*: $\vdash$ (*init w*) ⟶ ¬ ¬ (*init w*) **by** *auto*
hence *6*: $\vdash$ *df* (*init w*) ⟶ *df* (¬ ¬ (*init w*)) **by** (*rule DfImpDf*)
have *7*: $\vdash$ *df* (*init w*) ⟶ (*init w*) **using** *6 4* **using** *lift-imp-trans* **by** *metis*
have *8*: $\vdash$ (*init w*) ∧ *finite* ⟶ *df* (*init w*) **by** (*rule AndFiniteImpDf*)
from *7 8* **show** *?thesis*
by (*metis NowImpDiamond Prop10 StateAndChop df-d-def int-simps*(*17*) *inteq-reflection*
lift-and-com schop-d-def sometimes-d-def)
qed

lemma *StateSChop*:
$\vdash$ (*init w*)⌢ *f* ⟶ (*init w*)
by (*simp add*: *StateChopExportA schop-d-def*)

lemma *StateSChopExportA*:
$\vdash$ ((*init w*) ∧ *f*)⌢ *g* ⟶ (*init w*)
by (*meson AndSChopA StateSChop lift-imp-trans*)

lemma *StateAndSChop*:
$\vdash$ ((*init w*) ∧ *f*)⌢ *g* = ((*init w*) ∧ (*f*⌢ *g*))
by (*simp add*: *AndSChopB StateAndSChopImport StateSChopExportA Prop11 Prop12*)

lemma *StateAndSChopImpSChopRule*:
assumes $\vdash$ (*init w*) ∧ *f* ⟶ *f1*
shows $\vdash$ (*init w*) ∧ (*f*⌢ *g*) ⟶ (*f1*⌢ *g*)
proof −
have *1*: $\vdash$ (*init w*) ∧ *f* ⟶ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ ((*init w*) ∧ *f*)⌢ *g* ⟶ *f1*⌢ *g* **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash$ ((*init w*) ∧ *f*)⌢ *g* = ((*init w*) ∧ (*f*⌢ *g*)) **by** (*rule StateAndSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpSChopEqvSChop* :
assumes $\vdash$ (*init w*) ⟶ (*f* = *f1*)
shows $\vdash$ (*init w*) ⟶ ((*f*⌢ *g*) = (*f1*⌢ *g*))
proof −
have *1*: $\vdash$ (*init w*) ⟶ (*f*= *f1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) ∧ *f* ⟶ *f1* **by** *auto*
hence *3*: $\vdash$ (*init w*) ∧ (*f*⌢ *g*) ⟶ (*f1*⌢ *g*) **by** (*rule StateAndSChopImpSChopRule*)

have *4*: $\vdash (init\ w) \wedge f1 \longrightarrow f$ **using** *1* **by** *auto*
hence *5*: $\vdash (init\ w) \wedge (f1 \frown g) \longrightarrow (f \frown g)$ **by** (*rule StateAndSChopImpSChopRule*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvStateAndSChop*:
assumes $\vdash f = (init\ w) \wedge f1$
shows $\vdash (f \frown g) = ((init\ w) \wedge (f1 \frown g))$
proof −
have *1*: $\vdash f = ((init\ w) \wedge f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = (((init\ w) \wedge f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash ((init\ w) \wedge f1) \frown g = ((init\ w) \wedge (f1 \frown g))$ **by** (*rule StateAndSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfIntro*:
$\vdash f \wedge finite \longrightarrow df\ f$
proof −
have *1*: $\vdash finite \longrightarrow f \frown empty = f$ **by** (*rule SChopEmpty*)
have *2*: $\vdash empty \longrightarrow \#True$ **by** *auto*
hence *3*: $\vdash \Box(empty \longrightarrow \#True)$ **by** (*rule BoxGen*)
have *4*: $\vdash \Box(empty \longrightarrow \#True) \longrightarrow (f;\ empty \longrightarrow f;\ \#True)$ **by** (*rule BoxChopImpChop*)
have *5*: $\vdash f \frown empty \longrightarrow f \frown \#True$ **using** *3 4 MP* **by** (*simp add*: *RightSChopImpSChop*)
hence *6*: $\vdash f \frown empty \longrightarrow df\ f$ **by** (*simp add*: *df-d-def*)
from *1 6* **show** *?thesis* **using** *AndFiniteImpDf* **by** *blast*
qed

lemma *BfElim*:
$\vdash \quad bf\ f \wedge finite \longrightarrow f$
proof −
have *1*: $\vdash \neg f \wedge finite \longrightarrow df\ (\neg f)$ **by** (*rule DfIntro*)
have *2*: $\vdash (\neg f \wedge finite \longrightarrow df\ (\neg f)) \longrightarrow (\neg(df\ (\neg f)) \longrightarrow \neg(\neg\ f \wedge finite))$ **by** *simp*
have *21*: $\vdash \neg(\neg\ f \wedge finite) = (f \vee inf)$ **by** (*simp add*: *Valid-def finite-d-def*)
have *3*: $\vdash \neg\ (df\ (\neg f)) \longrightarrow f \vee inf$ **using** *1 2 21* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *Prop13 bf-d-def finite-d-def*)
qed

lemma *BfContraPosImpDist*:
$\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$
proof −
have *1*: $\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (df\ (\neg g)) \longrightarrow (df\ (\neg f))$ **by** (*rule BfImpDfImpDf*)
hence *2*: $\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (\neg(\ df\ (\neg f))) \longrightarrow (\neg(df\ (\neg g)))$ **by** *auto*
from *2* **show** *?thesis* **by** (*metis bf-d-def*)
qed

lemma *BfImpDist*:
$\vdash bf\ (f \longrightarrow g) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$
proof −

have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg\ g \longrightarrow \neg\ f)$ **by** *auto*
hence *2*: $\vdash \neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g)$ **by** *auto*
hence *3*: $\vdash bf\ (\neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g))$ **by** (*rule BfGen*)
have *4*: $\vdash bf\ (\neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g))$
$\longrightarrow$
$bf\ (f \longrightarrow g) \longrightarrow bf\ (\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BfContraPosImpDist*)
have *5*: $\vdash bf\ (f \longrightarrow g) \longrightarrow bf\ (\neg\ g \longrightarrow \neg\ f)$ **using** *3 4 MP* **by** *blast*
have *6*: $\vdash bf\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$ **by** (*rule BfContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FiniteImpBfImpBfRule*:
assumes $\vdash$ *finite* $\longrightarrow (f \longrightarrow g)$
shows $\vdash bf\ f \longrightarrow bf\ g$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow f \longrightarrow g$ **using** *assms* **by** *auto*
have *2*: $\vdash bf(f \longrightarrow g)$ **using** *1* **by** (*simp add*: *FiniteBfGen*)
have *3*: $\vdash\ bf(f \longrightarrow g) \longrightarrow bf\ f \longrightarrow bf\ g$ **using** *BfImpDist* **by** *blast*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteImpBfEqvRule*:
assumes $\vdash$ *finite* $\longrightarrow (f = g)$
shows $\vdash bf\ f = bf\ g$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow (f = g)$ **using** *assms* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow (f \longrightarrow g)$ **using** *1* **by** *auto*
have *3*: $\vdash bf\ f \longrightarrow bf\ g$ **by** (*simp add*: *2 FiniteImpBfImpBfRule*)
have *4*: $\vdash$ *finite* $\longrightarrow (g \longrightarrow f)$ **using** *1* **by** *auto*
have *5*: $\vdash bf\ g \longrightarrow bf\ f$ **by** (*simp add*: *4 FiniteImpBfImpBfRule*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *IfSChopEqvRule*:
assumes $\vdash f =\ if_i\ (init\ w)\ then\ f1\ else\ f2$
shows $\vdash f \frown g =\ if_i\ (init\ w)\ then\ (f1 \frown g)\ else\ (f2 \frown g)$
proof −
have *1*: $\vdash f =\ if_i\ (init\ w)\ then\ f1\ else\ f2$
using *assms* **by** *auto*
hence *2*: $\vdash f = (((init\ w) \wedge f1) \vee\ (\ (init\ (\neg\ w)) \wedge f2))$
unfolding *ifthenelse-d-def* **by** (*metis Initprop*(*2*) *int-eq*)
hence *3*: $\vdash f \frown g = (((init\ w) \wedge f1) \frown g \vee\ (\ \ (init\ (\neg\ w)) \wedge f2) \frown g)$
by (*rule OrSChopEqvRule*)
have *4*: $\vdash ((init\ w) \wedge f1) \frown g = ((init\ w) \wedge (f1 \frown g))$
by (*rule StateAndSChop*)
have *5*: $\vdash (\ \ (init\ (\neg\ w)) \wedge f2) \frown g =\ \ ((init\ (\neg\ w)) \wedge (f2 \frown g))$
by (*rule StateAndSChop*)
have *6*: $\vdash f \frown g = (((init\ w) \wedge f1 \frown g)\ \vee\ (\ \ (init\ (\neg\ w)) \wedge f2 \frown g))$

using *3 4 5* **by** *fastforce*
from *6* **show** *?thesis* **unfolding** *ifthenelse-d-def* **by** (*metis Initprop*(*2*) *inteq-reflection*)
qed

lemma *SChopOrEqvRule*:
assumes $\vdash g = (g1 \vee g2)$
shows $\vdash f \frown g = ((f \frown g1) \vee (f \frown g2))$
proof −
have *1*: $\vdash g = (g1 \vee g2)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = (f \frown (g1 \vee g2))$ **by** (*rule RightSChopEqvSChop*)
have *3*: $\vdash f \frown (g1 \vee g2) = (f \frown g1 \vee f \frown g2)$ **by** (*rule SChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopEqv*:
$\vdash (\ empty \vee f) \frown g = (g \vee (f \frown g))$
proof −
have *1*: $\vdash (\ empty \vee f) \frown g = ((\ empty \frown g) \vee (f \frown g))$ **by** (*rule OrSChopEqv*)
have *2*: $\vdash \ empty \frown g = g$ **by** (*rule EmptySChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqv*:
$\vdash (\ empty \vee \bigcirc f) \frown g = (g \vee \bigcirc(f \frown g))$
proof −
have *1*: $\vdash (\ empty \vee \bigcirc f) \frown g = (g \vee ((\bigcirc f) \frown g))$ **by** (*rule EmptyOrSChopEqv*)
have *2*: $\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$ **by** (*rule NextSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopImpRule*:
assumes $\vdash f \longrightarrow \ empty \vee f1$
shows $\vdash f \frown g \longrightarrow g \vee (f1 \frown g)$
proof −
have *1*: $\vdash f \longrightarrow \ empty \vee f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (\ empty \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash (\ empty \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (*rule EmptyOrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopEqvRule*:
assumes $\vdash f = (empty \vee f1)$
shows $\vdash f \frown g = (g \vee (f1 \frown g))$
proof −
have *1*: $\vdash f = (empty \vee f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = ((\ empty \vee f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (\ empty \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (*rule EmptyOrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopImpRule*:
assumes $\vdash f \longrightarrow \ empty \ \vee \ \bigcirc f1$
shows $\vdash f \frown g \longrightarrow g \vee \ \bigcirc(f1 \frown g)$
proof −
have *1*: $\vdash f \longrightarrow \ empty \ \vee \ \bigcirc f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (\ empty \ \vee \ \bigcirc f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash (\ empty \ \vee \ \bigcirc f1) \frown g = (g \vee \ \bigcirc(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqvRule*:
assumes $\vdash f = \ (empty \ \vee \ \bigcirc f1)$
shows $\vdash f \frown g = (g \vee \ \bigcirc(f1 \frown g))$
proof −
have *1*: $\vdash f = \ (empty \ \vee \ \bigcirc f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = ((\ empty \ \vee \ \bigcirc f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (\ empty \ \vee \ \bigcirc f1) \frown g = (g \vee \ \bigcirc(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopEmptyOrImpRule*:
assumes $\vdash \ g \longrightarrow \ empty \ \vee \ g1$
shows $\vdash f \frown g \wedge finite \longrightarrow f \vee \ (f \frown g1)$
proof −
have *1*: $\vdash g \longrightarrow \ empty \ \vee \ g1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (f \frown \ empty \ ) \vee \ (f \frown g1)$ **by** (*rule SChopOrImpRule*)
have *3*: $\vdash finite \longrightarrow f \frown \ empty \ = f$ **by** (*rule SChopEmpty*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateSChopBoxAndInfImpBox*:
$\vdash \Box \ (init \ w) \frown \Box \ (init \ w) \wedge inf \longrightarrow \Box \ (init \ w)$
by (*metis AndChopA BoxStateChopBoxEqvBox OrFiniteInf Prop03 int-eq lift-imp-trans schop-d-def*)

lemma *BoxStateSChopBoxEqvBox*:
$\vdash \ \ \Box \ (init \ w) \frown \Box \ (init \ w) = \Box \ (init \ w)$
proof −
have *1*: $\vdash (\Box \ (init \ w)) = ((init \ w) \wedge (\ empty \ \vee \ \bigcirc(\Box \ (init \ w))))$
by (*rule BoxEqvAndEmptyOrNextBox*)
hence *2*: $\vdash (\Box \ (init \ w) \frown \Box \ (init \ w)) =$
$((init \ w) \wedge ((\ empty \ \vee \ \bigcirc(\Box \ (init \ w))) \frown \Box \ (init \ w)))$
by (*metis StateAndSChop inteq-reflection*)
have *3*: $\vdash ((\ empty \ \vee \ \bigcirc(\Box \ (init \ w))) \frown \Box \ (init \ w)) =$
$(\Box \ (init \ w) \vee \ \bigcirc(\Box \ (init \ w) \frown \Box \ (init \ w)))$
by (*rule EmptyOrNextSChopEqv*)
have *4*: $\vdash (\Box \ (init \ w) \frown \Box \ (init \ w)) =$
$((init \ w) \wedge (\Box \ (init \ w) \vee \ \bigcirc(\Box \ (init \ w) \frown \Box \ (init \ w))))$
using *2 3* **by** *fastforce*
have *5*: $\vdash \neg \ (\Box \ (init \ w)) \longrightarrow \neg \ \ (init \ w) \vee \ \neg(\ \bigcirc(\Box \ (init \ w)))$
by (*rule NotBoxImpNotOrNotNextBox*)

have *6*: $\vdash (\Box\ (init\ w) \frown \Box\ (init\ w)) \wedge \neg(\ \Box\ (init\ w)) \longrightarrow$
$\bigcirc(\Box\ (init\ w) \frown \Box\ (init\ w)) \wedge \neg(\ \bigcirc(\Box\ (init\ w)))$
using *4 5* **by** *fastforce*
hence *7*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) \wedge finite \longrightarrow \Box\ (init\ w)$
by (*rule NextContra*)
have *8*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) \wedge inf \longrightarrow \Box\ (init\ w)$
by (*rule BoxStateSChopBoxAndInfImpBox*)
have *9*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) \wedge (finite \vee inf) \longrightarrow \Box\ (init\ w)$
using *7 8* **by** *fastforce*
hence *10*: $\vdash\ \Box\ (init\ w) \frown \Box\ (init\ w) \longrightarrow \Box\ (init\ w)$
using *FiniteOrInfinite* **by** *fastforce*
have *11*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge \Box\ (init\ w))$
by (*rule BoxEqvAndBox*)
have *12*: $\vdash empty \frown \Box\ (init\ w) = \Box\ (init\ w)$
by (*rule EmptySChop*)
have *13*: $\vdash ((init\ w) \wedge\ empty\ ) \frown \Box\ (init\ w) = ((init\ w) \wedge (\ empty \frown \Box\ (init\ w)))$
by (*rule StateAndSChop*)
have *14*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge\ empty\ ) \frown \Box\ (init\ w)$
using *11 12 13* **by** *fastforce*
have *15*: $\vdash (init\ w) \wedge\ empty\ \longrightarrow \Box\ (init\ w)$
by (*rule StateAndEmptyImpBoxState*)
hence *16*: $\vdash ((init\ w) \wedge\ empty\ ) \frown \Box\ (init\ w) \longrightarrow \Box\ (init\ w) \frown \Box\ (init\ w)$
by (*rule LeftSChopImpSChop*)
have *17*: $\vdash \Box\ (init\ w) \longrightarrow \Box\ (init\ w) \frown \Box\ (init\ w)$
using *14 16* **by** *fastforce*
from *10 17* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBoxStateImpBoxSYieldsNotBox*:
$\vdash\ \ \neg(\ \Box\ (init\ w)) \longrightarrow (\Box\ (init\ w))\ syields\ (\neg(\ \Box\ (init\ w)))$
proof −
have *1*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) = \Box\ (init\ w)$ **by** (*rule BoxStateSChopBoxEqvBox*)
have *2*: $\vdash \Box\ (init\ w) = (\neg\ \neg(\ \Box\ (init\ w)))$ **by** *auto*
hence *3*: $\vdash \Box\ (init\ w) \frown \Box(init\ w) = \Box(init\ w) \frown (\neg\neg(\Box(init\ w)))$ **by** (*rule RightSChopEqvSChop*)
have *4*: $\vdash \neg(\ \Box\ (init\ w)) \longrightarrow \neg\ (\Box\ (init\ w) \frown (\neg\ \neg\ (\Box\ (init\ w))))$ **using** *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *StateEqvBf*:
$\vdash\ \ (init\ w) = bf\ \ (init\ w)$
proof −
have *1*: $\vdash (init\ w) \longrightarrow bf\ \ (init\ w)$ **by** (*rule StateImpBf*)
have *2*: $\vdash bf\ \ (init\ w) \wedge finite \longrightarrow (init\ w)$ **by** (*rule BfElim*)
from *1 2* **show** *?thesis*
by (*metis* (*no-types*) *DfState Initprop*(*2*) *bf-d-def int-simps*(*4*) *inteq-reflection*)
qed

lemma *TrueSChopEqvDiamond*:
$\vdash\ \ \#True \frown f = \Diamond\ f$

using *DiamondSChopdef* **by** *fastforce*

lemma *BfAndEqvBfAndBf*:
$\vdash bf(f \wedge g) = (bf\ f \wedge bf\ g)$
proof −
have *1*: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
have *2*: $\vdash bf(f \wedge g) \longrightarrow bf\ f$ **by** (*simp add*: *1 BfImpBfRule*)
have *3*: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
have *4*: $\vdash bf(f \wedge g) \longrightarrow bf\ g$ **by** (*simp add*: *3 BfImpBfRule*)
have *5*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
have *6*: $\vdash bf\ f \longrightarrow bf\ (g \longrightarrow f \wedge g)$ **by** (*simp add*: *5 BfImpBfRule*)
have *7*: $\vdash bf\ (g \longrightarrow f \wedge g) \longrightarrow (bf\ g \longrightarrow bf(f \wedge g))$ **by** (*simp add*: *BfImpDist*)
have *8*: $\vdash bf\ f \wedge bf\ g \longrightarrow bf\ (f \wedge g)$ **using** *6 7* **by** *fastforce*
from *2 4 8* **show** *?thesis* **by** *fastforce*
qed

lemma *BfEqvBfImpAndBfImp*:
$\vdash bf(f = g) = (bf\ (f \longrightarrow g) \wedge bf(g \longrightarrow f))$
proof −
have *1*: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *auto*
have *2*: $\vdash bf(f = g) = bf((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (*simp add*: *1 BfEqvBf*)
have *3*: $\vdash bf((f \longrightarrow g) \wedge (g \longrightarrow f)) = (bf(f \longrightarrow g) \wedge bf(g \longrightarrow f))$ **by** (*simp add*: *BfAndEqvBfAndBf*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BfEqvImpSChopEqvSChop*:
$\vdash bf(f = f1) \longrightarrow f \frown g = f1 \frown g$
proof −
have *1*: $\vdash bf(f = f1) = (bf\ (f \longrightarrow f1) \wedge bf(f1 \longrightarrow f))$ **by** (*simp add*: *BfEqvBfImpAndBfImp*)
have *2*: $\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*simp add*: *BfSChopImpSChop*)
have *3*: $\vdash bf(f1 \longrightarrow f) \longrightarrow f1 \frown g \longrightarrow f \frown g$ **by** (*simp add*: *BfSChopImpSChop*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BfEqvDfEqvDf*:
$\vdash bf(f = g) \longrightarrow (df\ f = df\ g)$
proof −
have *1*: $\vdash bf(f = g) \longrightarrow (f \frown \#True) = (g \frown \#True)$
using *BfEqvImpSChopEqvSChop* **by** *fastforce*
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *FiniteImpEqvDfImpRule*:
assumes $\vdash$ *finite* $\longrightarrow f = g$
shows $\vdash df\ f = df\ g$

proof −
have *1*: ⊢ *finite* ⟶ *f* = *g* **using** *assms* **by** *auto*
have *2*: ⊢ *bf*(*f* = *g*) **using** *1* **by** (*simp add*: *FiniteBfGen*)
have *3*: ⊢ *bf*(*f* = *g*) ⟶ (*df f* = *df g*) **by** (*simp add*: *BfEqvDfEqvDf*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfEmpty*:
⊢ *df empty*
proof −
have *1*: ⊢ #*True* **by** *auto*
have *2*: ⊢ *empty* ⌢ #*True* = #*True* **by** (*rule EmptySChop*)
have *3*: ⊢ *empty* ⌢ #*True* **using** *1 2* **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *BfImpDf*:
⊢ *bf f* ⟶ *df f*
proof −
have *1*: ⊢ *f* ⟶ (*empty* ⟶ *f*) **by** *auto*
have *2*: ⊢ *bf f* ⟶ *bf*(*empty* ⟶ *f*) **by** (*simp add*: *1 BfImpBfRule*)
have *3*: ⊢ *bf*(*empty* ⟶ *f*) ⟶ *df empty* ⟶ *df f* **by** (*simp add*: *BfImpDfImpDf*)
have *4*: ⊢ *bf f* ⟶ *df empty* ⟶ *df f* **using** *2 3 lift-imp-trans* **by** *blast*
have *5*: ⊢ *df empty* **by** (*simp add*: *DfEmpty*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

3.4.5 Properties of SDa and SBa

lemma *SDaEqvDtDf*:
⊢ *sda f* = ◇ (*df f*)
proof −
have *1*: ⊢ #*True*⌢ (*f*⌢ #*True*) = #*True*⌢ (*f*⌢ #*True*) **by** *auto*
hence *2*: ⊢ #*True*⌢ (*f*⌢ #*True*) = #*True*⌢ *df f* **by** (*simp add*: *df-d-def*)
have *3*: ⊢ #*True*⌢ (*df f*) = ◇(*df f*) **by** (*simp add*: *TrueSChopEqvDiamond*)
have *4*: ⊢ #*True*⌢ (*f*⌢ #*True*) = ◇(*df f*) **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*sda-d-def*)
qed

lemma *SDaEqvDfDt*:
⊢ *sda f* = *df* (◇ *f*)
proof −
have *1*: ⊢ #*True*⌢ *f* = ◇ *f* **by** (*rule TrueSChopEqvDiamond*)
hence *2*: ⊢ (#*True*⌢ *f*)⌢ #*True* = (◇ *f*)⌢ #*True* **by** (*rule LeftSChopEqvSChop*)
hence *3*: ⊢ (#*True*⌢ *f*)⌢ #*True* = *df*(◇ *f*) **by** (*simp add*: *df-d-def*)
have *4*: ⊢ #*True*⌢ (*f*⌢ #*True*) = (#*True*⌢ *f*)⌢ #*True* **by** (*rule SChopAssoc*)
have *5*: ⊢ #*True*⌢ (*f*⌢ #*True*) = *df* (◇ *f*) **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *sda-d-def*)

qed

lemma *DtDfEqvDfDt*:
$\vdash \quad \Diamond\ (df\ \ f) =\ \ df\ (\Diamond\ f)$
by (*meson Prop04 SDaEqvDfDt SDaEqvDtDf*)

lemma *SBaEqvBfBt*:
$\vdash \quad sba\ f = bf(\ \Box\ f)$
proof −
have *1*: $\vdash\ \ sda\ (\neg\ \ f) =\ \ df(\ \Diamond\ (\neg\ \ f))$ **by** (*rule SDaEqvDfDt*)
have *2*: $\vdash \Diamond\ (\neg\ \ f) = (\neg(\ \Box\ f))$ **by** (*rule DiamondNotEqvNotBox*)
hence *3*: $\vdash\ \ df\ (\Diamond(\neg\ \ f)) =\ \ df\ (\neg\ (\Box\ f))$ **by** (*rule DfEqvDf*)
have *4*: $\vdash\ \ sda\ (\neg\ \ f) =\ \ df\ (\neg(\ \Box\ f))$ **using** *1 3* **by** *fastforce*
hence *5*: $\vdash (\neg\ \ (sda\ (\neg\ \ f))) = (\neg\ (\ df\ (\neg(\ \Box\ f))))$ **by** *auto*
hence *6*: $\vdash (\neg\ (\ sda\ (\neg\ \ f))) = bf(\ \Box\ f)$ **by** (*simp add*: *bf-d-def*)
from *6* **show** *?thesis* **by** (*simp add*: *sba-d-def*)
qed

lemma *DfNotEqvNotBf*:
$\vdash \quad df\ (\neg\ \ f) = (\neg(\ bf\ \ f))$
proof −
have *1*: $\vdash\ bf\ \ f = (\neg\ (\ df\ (\neg\ \ f)))$ **by** (*simp add*: *bf-d-def*)
from *1* **show** *?thesis* **by** *auto*
qed

lemma *DfDfNotEqvNotBfBf*:
$\vdash\ df\ (df\ (\neg\ f)) = (\neg\ (bf\ \ (bf\ f)))$
proof −
have *1*: $\vdash\ df\ (\neg\ f) = (\neg\ bf\ f)$ **by** (*simp add*: *DfNotEqvNotBf*)
have *2*: $\vdash\ df\ (\ df\ (\neg\ f)) = df\ (\neg\ bf\ f)$ **by** (*simp add*: *1 DfEqvDf*)
have *3*: $\vdash\ df\ (\neg\ bf\ f) = (\neg\ bf\ (bf\ f))$ **by** (*simp add*: *DfNotEqvNotBf*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfDtEqvDtDf*:
$\vdash\ df(\Diamond\ f) = \Diamond(df\ f)$
proof −
have *1*: $\vdash (\#True^\frown f)^\frown \#True = \#True^\frown(f^\frown \#True)$
using *SChopAssoc* **by** *fastforce*
have *2*: $\vdash (\Diamond\ f)^\frown \#True = \Diamond(f^\frown \#True)$
using *1* **by** (*metis TrueSChopEqvDiamond int-eq*)
from *1 2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfDtNotEqvNotBfBt*:
$\vdash\ df(\Diamond(\neg\ f)) = (\neg(bf(\Box\ f)))$

proof −
have *1*: ⊢ ◇ (¬*f*) = (¬ (□ *f*)) **by** (*simp add*: *DiamondNotEqvNotBox*)
have *2*: ⊢ *df*(◇ (¬*f*)) = *df*(¬ (□ *f*)) **by** (*simp add*: *1 DfEqvDf*)
have *3*: ⊢ *df*(¬(□ *f*)) = (¬(*bf* (□ *f*))) **by** (*simp add*: *DfNotEqvNotBf*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DtDfNotEqvNotBtBf*:
⊢ ◇(*df* (¬ *f*)) = (¬ (□(*bf f*)))
proof −
have *1*: ⊢ *df*(¬ *f*) = (¬ (*bf f*)) **using** *DfNotEqvNotBf* **by** *blast*
have *2*: ⊢ ◇(*df*(¬ *f*)) = ◇(¬ (*bf f*)) **by** (*simp add*: *1 DiamondEqvDiamond*)
have *3*: ⊢ ◇(¬ (*bf f*)) = (¬ □(*bf f*)) **by** (*simp add*: *DiamondNotEqvNotBox*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaEqvBtBf*:
⊢ *sba f* = □ (*bf f*)
proof −
have *1*: ⊢ *sda* (¬ *f*) = ◇ (*df* (¬ *f*)) **by** (*rule SDaEqvDtDf*)
have *2*: ⊢ *df* (¬ *f*) = (¬ (*bf f*)) **by** (*rule DfNotEqvNotBf*)
hence *3*: ⊢ ◇ (*df* (¬ *f*)) = ◇(¬ (*bf f*)) **by** (*rule DiamondEqvDiamond*)
have *4*: ⊢ (¬ (◇(¬ (*bf f*)))) = □(*bf f*) **by** (*rule NotDiamondNotEqvBox*)
have *5*: ⊢ (¬ (*sda* (¬ *f*))) = □(*bf f*) **using** *1 2 3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *sba-d-def*)
qed

lemma *BaImpSBa*:
⊢ *ba f* ⟶ *sba f*
using *BaEqvBiBt BiImpBf SBaEqvBfBt* **by** *fastforce*

lemma *SDaImpDa*:
⊢ *sda f* ⟶ *da f*
proof −
have *1*: ⊢ *ba* (¬ *f*) ⟶ *sba* (¬*f*)
using *BaImpSBa* **by** *blast*
have *2*: ⊢ ¬ *sba* (¬*f*) ⟶ ¬ *ba* (¬ *f*)
using *1* **by** *fastforce*
from *2* **show** *?thesis* **by** (*simp add*: *sba-d-def ba-d-def*)
qed

lemma *BtBfEqvBfBt*:
⊢ □ (*bf f*) = *bf*(□ *f*)
proof −
have *1*: ⊢ *sba f* = □ (*bf f*) **by** (*rule SBaEqvBtBf*)
have *2*: ⊢ *sba f* = *bf*(□ *f*) **by** (*rule SBaEqvBfBt*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvSBaBoxState*:
$\vdash \quad \Box\ (init\ w) = \ \ sba\ (\Box\ (init\ w))$
proof −
have *1*: $\vdash (init\ w) = bf\ \ (init\ w)$ **by** (*rule StateEqvBf*)
hence *2*: $\vdash \Box\ (init\ w) = \Box\ (bf\ \ (init\ w))$ **by** (*rule BoxEqvBox*)
have *3*: $\vdash \Box\ (bf\ \ (init\ w)) = bf(\ \Box\ (init\ w))$ **by** (*rule BtBfEqvBfBt*)
have *4*: $\vdash \Box\ (init\ w) = \Box(\Box\ (init\ w))$ **by** (*rule BoxEqvBoxBox*)
hence *5*: $\vdash bf(\ \Box\ (init\ w)) = bf\ (\Box(\Box\ (init\ w)))$ **by** (*rule BfEqvBf*)
have *6*: $\vdash \ \ sba(\ \Box\ (init\ w)) = bf(\ \Box(\Box\ (init\ w)))$ **by** (*rule SBaEqvBfBt*)
from *2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpBf*:
$\vdash \quad sba\ \ f \longrightarrow bf\ \ f$
proof −
have *1*: $\vdash \ \ sba\ \ f = \Box(bf\ \ f)$ **by** (*rule SBaEqvBtBf*)
have *2*: $\vdash \Box(bf\ \ f) \longrightarrow bf\ \ f$ **by** (*rule BoxElim*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *BaImpBf*:
$\vdash \quad ba\ \ f \longrightarrow bf\ \ f$
proof −
have *1*: $\vdash \ \ ba\ \ f = \Box(bi\ \ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash \Box(bi\ \ f) \longrightarrow bi\ \ f$ **by** (*rule BoxElim*)
have *3*: $\vdash bi\ \ f \longrightarrow bf\ f$ **by** (*simp add*: *BiImpBf*)
from *1 2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *SBaImpBt*:
$\vdash \quad sba\ \ f \wedge finite \longrightarrow \Box\ f$
proof −
have *1*: $\vdash \ \ sba\ \ f = bf(\ \Box\ f)$ **by** (*rule SBaEqvBfBt*)
have *2*: $\vdash bf(\ \Box\ f) \wedge finite \longrightarrow \Box\ f$ **by** (*rule BfElim*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondImpSDa*:
$\vdash \quad \Diamond\ f \wedge finite \longrightarrow \ \ sda\ \ f$
by (*metis AndFiniteImpDf SDaEqvDfDt inteq-reflection*)

lemma *DfImpSDa*:
$\vdash \quad df\ \ f \longrightarrow \ \ sda\ \ f$
using *NowImpDiamond SDaEqvDtDf* **by** *fastforce*

lemma *BoxAndSChopImport*:
$\vdash \quad \Box\ h \wedge f \frown g \longrightarrow f \frown (h \wedge g)$
proof −
have *1*: $\vdash h \longrightarrow g \longrightarrow (h \wedge g)$ **by** *auto*

hence *2*: $\vdash \Box\ h \longrightarrow \Box(g \longrightarrow (h \wedge g))$ **by** (*rule ImpBoxRule*)
have *3*: $\vdash \Box(g \longrightarrow (h \wedge g)) \longrightarrow f \frown g \longrightarrow f \frown (h \wedge g)$ **by** (*rule BoxSChopImpSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaAndSChopImport*:
$\vdash$ *sba f* $\wedge$ *finite* $\wedge$ $(g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof −
have *1*: $\vdash$ *sba f* $\longrightarrow$ *bf f* **by** (*rule SBaImpBf*)
have *2*: $\vdash$ *bf f* $\wedge$ $(g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (*rule BfAndSChopImport*)
have *3*: $\vdash$ *sba f* $\wedge$ *finite* $\longrightarrow \Box\ f$ **by** (*rule SBaImpBt*)
have *4*: $\vdash \Box\ f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (*rule BoxAndSChopImport*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndSChopImport*:
$\vdash$ *ba f* $\wedge$ $(g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof −
have *1*: $\vdash$ *ba f* $\longrightarrow$ *bi f* **by** (*rule BaImpBi*)
have *2*: $\vdash$ *bi f* $\wedge$ $(g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (*rule BiAndSChopImport*)
have *3*: $\vdash$ *ba f* $\longrightarrow \Box\ f$ **by** (*rule BaImpBt*)
have *4*: $\vdash \Box\ f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (*rule BoxAndSChopImport*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndCommute*:
$\vdash f \frown (g \wedge g1) = f \frown (g1 \wedge g)$
proof −
have *1*: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightSChopEqvSChop*)
qed

lemma *SChopAndA*:
$\vdash f \frown (g \wedge g1) \longrightarrow f \frown g$
proof −
have *1*: $\vdash (g \wedge g1) \longrightarrow g$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightSChopImpSChop*)
qed

lemma *SChopAndB*:
$\vdash f \frown (g \wedge g1) \longrightarrow f \frown g1$
proof −
have *1*: $\vdash (g \wedge g1) \longrightarrow g1$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightSChopImpSChop*)
qed

lemma *BoxStateAndSChopEqvSChop*:
$\vdash (\Box\ (init\ w) \wedge finite \wedge (f \frown g)) = ((\Box\ (init\ w) \wedge f) \frown (\Box\ (init\ w) \wedge g) \wedge finite)$
proof −
have *1*: $\vdash \Box\ (init\ w) =$ *sba*($\Box\ (init\ w)$)

by (*rule BoxStateEqvSBaBoxState*)
have *2*: $\vdash$ *sba*($\Box$ (*init w*)) $\wedge$ *finite* $\wedge$ ($f\frown$ *g*) $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *f*)$\frown$ ($\Box$ (*init w*) $\wedge$ *g*)
by (*rule SBaAndSChopImport*)
have *3*: $\vdash$ $\Box$ (*init w*) $\wedge$ *finite* $\wedge$ ($f\frown$ *g*) $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *f*)$\frown$ ($\Box$ (*init w*) $\wedge$ *g*)
using *1 2* **by** *fastforce*
have *11*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*)$\frown$ ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ ($\Box$ (*init w*))$\frown$ ($\Box$ (*init w*) $\wedge$ *g*)
by (*rule AndSChopA*)
have *12*: $\vdash$ ($\Box$ (*init w*))$\frown$ ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ ($\Box$ (*init w*))$\frown$ ($\Box$ (*init w*))
by (*rule SChopAndA*)
have *13*: $\vdash$ ($\Box$ (*init w*))$\frown$ ($\Box$ (*init w*)) = $\Box$ (*init w*)
by (*rule BoxStateSChopBoxEqvBox*)
have *14*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*)$\frown$ ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ *f*$\frown$ ($\Box$ (*init w*) $\wedge$ *g*)
by (*rule AndSChopB*)
have *15*: $\vdash$ *f*$\frown$ ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ *f*$\frown$ *g*
by (*rule SChopAndB*)
have *16*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*)$\frown$ ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ $\Box$ (*init w*) $\wedge$ ($f\frown$ *g*)
using *11 12 13 14 15* **by** *fastforce*
from *3 16* **show** *?thesis* **by** *fastforce*
qed

lemma *DfEqvNotBfNot*:
$\vdash$ *df f* = (¬(*bf* (¬ *f*)))
proof −
have *1*: $\vdash$ *bf* (¬ *f*) = (¬ (*df* (¬ ¬ *f*))) **by** (*simp add*: *bf-d-def*)
hence *2*: $\vdash$ *df* (¬ ¬ *f*) = (¬(*bf* (¬ *f*))) **by** *auto*
have *3*: $\vdash$ *f* = (¬ ¬ *f*) **by** *auto*
hence *4*: $\vdash$ *df f* = *df* (¬ ¬ *f*) **by** (*rule DfEqvDf*)
from *2 4* **show** *?thesis* **by** *auto*
qed

lemma *SChopAndBoxImport*:
$\vdash$ $f\frown$ *g* $\wedge$ $\Box$ *h* $\longrightarrow$ *f*$\frown$ (*g* $\wedge$ *h*)
proof −
have *1*: $\vdash$ $\Box$ *h* $\wedge$ *f*$\frown$ *g* $\longrightarrow$ *f*$\frown$ (*h* $\wedge$ *g*) **by** (*rule BoxAndSChopImport*)
have *2*: $\vdash$ *f*$\frown$ (*h* $\wedge$ *g*) = *f*$\frown$ (*g* $\wedge$ *h*) **by** (*rule SChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *AndSChopAndCommute*:
$\vdash$ (*f* $\wedge$ *g*)$\frown$ (*f1* $\wedge$ *g1*) = (*g* $\wedge$ *f*)$\frown$ (*g1* $\wedge$ *f1*)
proof −
have *1*: $\vdash$ (*f* $\wedge$ *g*)$\frown$ (*f1* $\wedge$ *g1*) = (*g* $\wedge$ *f*)$\frown$ (*f1* $\wedge$ *g1*) **by** (*rule AndSChopCommute*)
have *2*: $\vdash$ (*g* $\wedge$ *f*)$\frown$ (*f1* $\wedge$ *g1*) = (*g* $\wedge$ *f*)$\frown$ (*g1* $\wedge$ *f1*) **by** (*rule SChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopImpSChop*:
assumes $\vdash$ *f* $\longrightarrow$ *f1*
$\vdash$ *g* $\longrightarrow$ *g1*

shows $\vdash f \frown g \longrightarrow f1 \frown g1$
proof −
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f1 \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence *4*: $\vdash f1 \frown g \longrightarrow f1 \frown g1$ **by** (*rule RightSChopImpSChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopEqvSChop*:
assumes $\vdash f = f1$
$\vdash g = g1$
shows $\vdash f \frown g = f1 \frown g1$
proof −
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = f1 \frown g$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash g = g1$ **using** *assms* **by** *auto*
hence *4*: $\vdash f1 \frown g = f1 \frown g1$ **by** (*rule RightSChopEqvSChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxSChopImpSChopBox*:
$\vdash \quad \Box\, h \longrightarrow f \frown g \longrightarrow f \frown (\Box\, h \wedge g)$
proof −
have *1*: $\vdash \Box\, h \longrightarrow \Box(g \longrightarrow \Box\, h \wedge g\,)$ **by** (*rule BoxImpBoxImpBox*)
have *2*: $\vdash \Box(g \longrightarrow \Box\, h \wedge g\,) \longrightarrow f \frown g \longrightarrow f \frown (\Box\, h \wedge g)$ **by** (*rule BoxSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotChopEqvSYieldsNot*:
$\vdash \quad (\neg\, (f \frown g)) = f\ syields\ \ (\neg\ \ g)$
proof −
have *1*: $\vdash g = (\neg\, \neg\ \ g)$ **by** *auto*
hence *2*: $\vdash f \frown g = f \frown (\neg\, \neg\ \ g)$ **by** (*rule RightSChopEqvSChop*)
hence *3*: $\vdash (\neg\, (f \frown g)) = (\neg\, (f \frown (\neg\, \neg\ \ g)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *NotDfFalse*:
$\vdash \quad \neg\, (\ df\ \ \#False)$
proof −
have *1*: $\vdash (init\ \#True) \longrightarrow bf\ (init\ \#True)$ **by** (*rule StateImpBf*)
hence *2*: $\vdash \#True \longrightarrow bf\ \#True$ **by** (*simp add*: *BfGen*)
have *3*: $\vdash \#True$ **by** *auto*
have *4*: $\vdash bf\ \#True$ **using** *2 3 MP* **by** *auto*
hence *5*: $\vdash \neg\, (\ df\ (\neg\, \#True))$ **by** (*simp add*: *bf-d-def*)
have *6*: $\vdash (\neg\, \#True) = \ \#False$ **by** *auto*
hence *7*: $\vdash \ df\ (\neg\, \#True) = \ df\ \ \#False$ **by** (*rule DfEqvDf*)
from *5 7* **show** *?thesis* **by** *auto*
qed

lemma *StateAndEmptySChop*:
$\vdash \quad ((init\ w) \wedge \ empty\ ) \frown f = ((init\ w) \wedge f)$
proof −
have *1*: $\vdash ((init\ w) \wedge \ empty\ ) \frown f = ((init\ w) \wedge \ empty \frown f)$ **by** (*rule StateAndSChop*)
have *2*: $\vdash \ empty \frown f = f$ **by** (*rule EmptySChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndNextSChop*:
$\vdash \quad ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge \bigcirc(f \frown g))$
proof −
have *1*: $\vdash ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge (\bigcirc f) \frown g)$ **by** (*rule StateAndSChop*)
have *2*: $\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$ **by** (*rule NextSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NextStateAndSChop*:
$\vdash \quad \bigcirc(((init\ w) \wedge f) \frown g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f \frown g))$
proof −
have *1*: $\vdash ((init\ w) \wedge f) \frown g = ((init\ w) \wedge f \frown g)$ **by** (*rule StateAndSChop*)
hence *2*: $\vdash \bigcirc(((init\ w) \wedge f) \frown g) = \bigcirc((init\ w) \wedge f \frown g)$ **by** (*rule NextEqvNext*)
have *3*: $\vdash \bigcirc((init\ w) \wedge f \frown g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f \frown g))$ **by** (*rule NextAndEqvNextAndNext*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateSYieldsEqv*:
$\vdash \quad ((init\ w) \longrightarrow (f\ syields\ \ g)) = ((init\ w) \wedge f)\ syields\ \ g$
proof −
have *1*: $\vdash ((init\ w) \wedge f) \frown (\neg\ \ g) = ((init\ w) \wedge f \frown (\neg\ \ g))$ **by** (*rule StateAndSChop*)
hence *2*: $\vdash ((init\ w) \longrightarrow \neg\ (f \frown (\neg\ \ g))) = (\neg\ (((init\ w) \wedge f) \frown (\neg\ \ g)\ ))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *StateAndDf*:
$\vdash \quad ((init\ w) \wedge \ df\ \ f) = \ df\ ((init\ w) \wedge f)$
proof −
have *1*: $\vdash ((init\ w) \wedge f) \frown \#True = \ ((init\ w) \wedge f \frown \#True)$ **by** (*rule StateAndSChop*)
from *1* **show** *?thesis* **by** (*metis df-d-def inteq-reflection*)
qed

lemma *DfNext*:
$\vdash \quad df(\ \bigcirc f) = \bigcirc\ (df\ \ f)$
proof −
have *1*: $\vdash (\bigcirc f) \frown \#True = \bigcirc(f \frown \#True)$ **by** (*rule NextSChop*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfNextState*:
$\vdash \quad df(\bigcirc\,(init\ w)) = \bigcirc\,(init\ w)$
proof −
have *1*: $\vdash \ df(\bigcirc\,(init\ w)) = \bigcirc(\ df\ \ (init\ w))$ **by** (*rule DfNext*)
have *2*: $\vdash \ df\ \ (init\ w) = (init\ w)$ **by** (*rule DfState*)
hence *3*: $\vdash \bigcirc(\ df\ \ (init\ w)) = \bigcirc\,(init\ w)$ **by** (*rule NextEqvNext*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfStateAndNextStateEqvStateAndNextState*:
$\vdash df(init\ w \wedge \bigcirc(init\ w1)) = (init\ w \wedge \bigcirc(init\ w1))$
proof −
have *1*: $\vdash (init\ w \wedge \bigcirc(init\ w1)) \frown \#True = (\ init\ w \wedge \bigcirc((init\ w1) \frown \#True))$
using *StateAndNextSChop* **by** *blast*
have *2*: $\vdash df(init\ w \wedge \bigcirc(init\ w1)) = (\ init\ w \wedge \bigcirc((init\ w1) \frown \#True))$
using *1* **by** (*simp add*: *df-d-def*)
have *3*: $\vdash df(init\ w1) = init\ w1$
by (*simp add*: *DfState*)
have *4*: $\vdash skip \frown df(init\ w1) = skip \frown (init\ w1)$
by (*simp add*: *3 RightSChopEqvSChop*)
have *5*: $\vdash \bigcirc(df(init\ w1)) = \bigcirc(init\ w1)$
by (*simp add*: *3 NextEqvNext*)
from *2 5* **show** *?thesis* **by** (*metis df-d-def int-eq*)
qed

lemma *StateImpBfGen*:
assumes $\vdash (init\ w) \longrightarrow f$
shows $\vdash (init\ w) \longrightarrow bf\ \ f$
proof −
have *1*: $\vdash (init\ w) \longrightarrow f$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ \ f \longrightarrow \neg\ \ (init\ w)$ **by** *auto*
hence *3*: $\vdash \ \ df\ (\neg\ \ f) \longrightarrow \ \ df\ (\neg\ \ (init\ w))$ **by** (*rule DfImpDf*)
hence *4*: $\vdash \ \ df\ (\neg\ \ f) \longrightarrow \ \ df\ \ (init\ (\neg w))$ **by** (*metis Initprop*(*2*) *inteq-reflection*)
have *5*: $\vdash \ \ df\ (init\ (\neg\ w)) = \ \ \ (init\ (\neg\ w))$ **by** (*rule DfState*)
have *6*: $\vdash \ \ df\ (\neg\ \ f) \longrightarrow \neg\ \ (init\ w)$ **using** *4 5* **using** *Initprop*(*2*) **by** *fastforce*
hence *7*: $\vdash (init\ w) \longrightarrow \neg\ (\ df\ (\neg\ \ f))$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

lemma *SChopAndNotSChopImp*:
$\vdash \quad f \frown g \wedge \neg\ (f \frown g1) \longrightarrow f \frown (g \wedge \neg\ \ g1)$
proof −
have *1*: $\vdash g \longrightarrow (g \wedge \neg\ \ g1) \vee \ \ g1$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown ((g \wedge \neg\ \ g1) \vee \ \ g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown ((g \wedge \neg\ \ g1) \vee \ \ g1) \longrightarrow (f \frown (g \wedge \neg\ \ g1)) \vee \ \ (f \frown g1)$ **by** (*rule SChopOrImp*)
have *4*: $\vdash f \frown g \longrightarrow f \frown (g \wedge \neg\ \ g1) \vee \ \ f \frown g1$ **using** *2 3 MP* **by** *fastforce*
from *4* **show** *?thesis* **by** *auto*
qed

lemma *SChopAndSYieldsImp*:
$\vdash \quad f \frown g \wedge f$ *syields* $g1 \longrightarrow f \frown (g \wedge g1)$
proof −
have *1*: $\vdash g \longrightarrow (g \wedge g1) \vee \neg g1$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown ((g \wedge g1) \vee \neg g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown ((g \wedge g1) \vee \neg g1) \longrightarrow (f \frown (g \wedge g1)) \vee (f \frown (\neg g1))$ **by** (*rule SChopOrImp*)
have *4*: $\vdash f \frown g \longrightarrow f \frown (g \wedge g1) \vee f \frown (\neg g1)$ **using** *2 3 MP* **by** *fastforce*
hence *5*: $\vdash f \frown g \wedge \neg (f \frown (\neg g1)) \longrightarrow f \frown (g \wedge g1)$ **by** *auto*
from *5* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SChopAndSYieldsMP*:
$\vdash \quad f \frown g \wedge f$ *syields* $(g \longrightarrow g1) \longrightarrow f \frown g1$
proof −
have *1*: $\vdash f \frown g \wedge f$ *syields* $(g \longrightarrow g1) \longrightarrow f \frown (g \wedge (g \longrightarrow g1))$ **by** (*rule SChopAndSYieldsImp*)
have *2*: $\vdash g \wedge (g \longrightarrow g1) \longrightarrow g1$ **by** *auto*
hence *3*: $\vdash f \frown (g \wedge (g \longrightarrow g1)) \longrightarrow f \frown g1$ **by** (*rule RightSChopImpSChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *OrSYieldsImp*:
$\vdash \quad (f \vee f1)$ *syields* $g = ((f$ *syields* $g) \wedge (f1$ *syields* $g))$
proof −
have *1*: $\vdash ((f \vee f1) \frown (\neg g)) = ((f \frown (\neg g)) \vee (f1 \frown (\neg g)))$ **by** (*rule OrSChopEqv*)
hence *2*: $\vdash (\neg ((f \vee f1) \frown (\neg g))) = (\neg (f \frown (\neg g)) \wedge \neg (f1 \frown (\neg g)))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *LeftSYieldsImpSYields*:
assumes $\vdash f \longrightarrow f1$
shows $\vdash (f1$ *syields* $g) \longrightarrow (f$ *syields* $g)$
proof −
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown (\neg g) \longrightarrow f1 \frown (\neg g)$ **by** (*rule LeftSChopImpSChop*)
hence *3*: $\vdash \neg (f1 \frown (\neg g)) \longrightarrow \neg (f \frown (\neg g))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *LeftSYieldsEqvSYields*:
assumes $\vdash f = f1$
shows $\vdash (f$ *syields* $g) = (f1$ *syields* $g)$
proof −
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown (\neg g) = f1 \frown (\neg g)$ **by** (*rule LeftSChopEqvSChop*)
hence *3*: $\vdash (\neg (f \frown (\neg g))) = (\neg (f1 \frown (\neg g)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

3.4.6 Properties of SFin

lemma *SFinEqvTrueSChopAndEmpty*:
$\vdash$ *sfin f* = #*True*⌢(*f* ∧ *empty*)
proof −
have *01*: ⊢ *sfin f* = (¬ *fin* (¬ *f*))
by (*simp add*: *sfin-d-def*)
have *02*:⊢ (¬ *fin* (¬ *f*)) = (¬ (□ (*empty* ⟶ ¬ *f*)))
by (*simp add*: *fin-d-def*)
have *03*:⊢ (¬ (□ (*empty* ⟶ ¬ *f*))) = ◇(¬(*empty* ⟶ ¬ *f*))
by (*simp add*: *always-d-def*)
have *04*: ⊢ ¬(*empty* ⟶ ¬ *f*) = (*empty* ∧ *f*)
by *auto*
have *05*:⊢ ◇(¬(*empty* ⟶ ¬ *f*)) = ◇ (*empty* ∧ *f*)
using *04 inteq-reflection* **by** *fastforce*
from *01 02 03 05* **show** *?thesis*
by (*metis SChopAndCommute TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *DiamondSFin*:
⊢ ◇(*sfin w*) = *sfin w*
by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopFiniteEqvFinite FiniteOr FiniteOrInfinite InfEqvNotFinite OrFiniteInf SFinEqvTrueSChopAndEmpty finite-d-def int-eq-true int-simps*(*21*) *inteq-reflection schop-d-def sometimes-d-def*)

lemma *SChopSFinExportA*:
⊢ *f*⌢(*g* ∧ *sfin w*) ⟶ *sfin w*
using *DiamondSFin*
by (*metis SChopAndB SChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *SFinImpBox*:
⊢ *sfin w* ⟶ □(*sfin w*)
by (*metis* (*mono-tags*, *lifting*) *DiamondFin always-d-def intI int-eq int-simps*(*4*) *sfin-d-def unl-lift2*)

lemma *SFinAndSChopImport*:
⊢ (*sfin w*) ∧ (*f*⌢*g*) ⟶ *f*⌢((*sfin w*) ∧ *g*)
proof −
have *1*: ⊢ *sfin w* ⟶ □(*sfin w*) **by** (*rule SFinImpBox*)
hence *2*: ⊢ *sfin w* ∧ (*f*⌢*g*) ⟶ □(*sfin w*) ∧ (*f*⌢*g*) **by** *auto*
have *3*: ⊢ □(*sfin w*) ∧ (*f*⌢*g*) ⟶ *f*⌢((*sfin w*) ∧ *g*) **using** *BoxAndSChopImport* **by** *blast*
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SFinAndSChop*:
⊢ (*f*⌢(*g* ∧ *sfin w*)) = (*sfin w* ∧ *f*⌢*g*)
using *SFinAndSChopImport SChopSFinExportA SChopAndA SChopAndCommute*
by *fastforce*

lemma *SChopAndEmptyEqvEmptySChopEmpty*:
⊢ ((*f*⌢*g*) ∧ *empty*) = (*f* ∧ *empty*)⌢(*g* ∧ *empty*)
by (*auto simp*: *itl-defs zero-enat-def*)

lemma *SFinAndEmpty*:
$\vdash ((sfin\ w) \wedge empty) = (w \wedge empty)$
proof −
have *1*: $\vdash ((sfin\ w) \wedge empty) = (\#True \frown (w \wedge empty) \wedge empty)$
using *SFinEqvTrueSChopAndEmpty* **by** *fastforce*
have *2*: $\vdash (\#True \frown (w \wedge empty) \wedge empty) = ((\#True \wedge empty) \frown (w \wedge empty))$
by (*simp add*: *FiniteChopAndEmptyEqvChopAndEmpty schop-d-def*)
have *3*: $\vdash (\#True \wedge empty) \frown (w \wedge empty) = (empty \frown (w \wedge empty))$
using *LeftSChopEqvSChop* **by** *fastforce*
have *4*: $\vdash (empty \frown (w \wedge empty)) = (w \wedge empty)$
using *EmptySChop* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *AndSFinEqvSChopAndEmpty*:
$\vdash \quad ((f \wedge finite) \wedge sfin\ g) = f \frown (g \wedge \ empty\ )$
proof −
have *1*: $\vdash ((f \wedge finite) \wedge sfin\ g) = (f \frown empty \wedge sfin\ g)$
using *SChopEmpty*
by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FiniteImpAnd Prop10 SChopImpDiamond inteq-reflection lift-and-com*)
have *2*: $\vdash (sfin\ g \wedge f \frown empty) = (f \frown (empty \wedge sfin\ g))$
using *SFinAndSChop* **by** *fastforce*
have *3*: $\vdash (empty \wedge sfin\ g) = \ (sfin\ g \wedge empty)$
by *auto*
have *4*: $\vdash (sfin\ g \wedge empty) = (g \wedge empty)$
using *SFinAndEmpty* **by** *metis*
have *5*: $\vdash (empty \wedge sfin\ g) = (g \wedge empty)$
using *3 4* **by** *auto*
hence *6*: $\vdash f \frown (empty \wedge sfin\ g) = f \frown (g \wedge empty)$
using *RightSChopEqvSChop* **by** *blast*
from *1 2 5* **show** *?thesis* **by** (*metis inteq-reflection lift-and-com*)
qed

lemma *AndSFinEqvSChopStateAndEmpty*:
$\vdash \quad ((f \wedge finite) \wedge \ sfin\ \ (init\ w)) = f \frown ((init\ w) \wedge \ \ empty\ )$
using *AndSFinEqvSChopAndEmpty* **by** *blast*

lemma *DiamondEqvEmptyOrNextDiamond*:
$\vdash \Diamond f = (f \ \vee \bigcirc(\Diamond f))$
proof −
have *1*: $\vdash \Box (\neg f) = ((\neg f) \wedge wnext(\Box (\neg f)))$
by (*simp add*: *BoxEqvAndWnextBox*)
have *2*: $\vdash (\neg \Diamond f) = ((\neg f) \wedge wnext(\Box (\neg f)))$
using *1* **by** (*simp add*: *always-d-def*)
have *3*: $\vdash \ \Diamond f = (f \vee \neg(wnext(\Box (\neg f))))$
using *2* **by** *auto*
have *4*: $\vdash (\neg(wnext(\Box (\neg f)))) = \bigcirc(\neg\Box(\neg f))$
by (*simp add*: *wnext-d-def*)

have *5*: $\vdash \neg\Box(\neg f) = \Diamond f$
by (*simp add*: *always-d-def*)
have *6*: $\vdash \bigcirc(\neg\Box(\neg f)) = \bigcirc(\Diamond f)$
using *5* **using** *inteq-reflection* **by** *force*
from *3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinStateEqvStateAndEmptyOrNextSFinState*:
$\vdash$ *sfin* (*init w*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*sfin* (*init w*)))
proof −
have *01*: $\vdash$ *sfin* (*init w*) = #*True*$\frown$((*init w*) $\wedge$ *empty*)
by (*simp add*: *SFinEqvTrueSChopAndEmpty*)
have *02*: $\vdash$ #*True*$\frown$((*init w*) $\wedge$ *empty*) = $\Diamond$ ((*init w*) $\wedge$ *empty*)
by (*simp add*: *TrueSChopEqvDiamond*)
have *03*: $\vdash$ $\Diamond$ ((*init w*) $\wedge$ *empty*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*sfin* (*init w*)))
using *DiamondEqvEmptyOrNextDiamond 02 01* **by** (*metis inteq-reflection*)
from *01 02 03* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSChopEqvOr*:
$\vdash$ (*sfin* (*init w*))$\frown$ *f* = (((*init w*) $\wedge$ *f*) $\vee$ $\bigcirc$((*sfin* (*init w*))$\frown$ *f*))
proof −
have *1*: $\vdash$ *sfin* (*init w*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*sfin* (*init w*)))
by (*rule SFinStateEqvStateAndEmptyOrNextSFinState*)
hence *2*: $\vdash$ (*sfin* (*init w*))$\frown$ *f* = (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$(*sfin* (*init w*)))$\frown$ *f*
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash$ (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$ (*sfin* (*init w*)))$\frown$ *f*
= (((*init w*) $\wedge$ *empty*)$\frown$ *f* $\vee$ ($\bigcirc$ (*sfin* (*init w*)))$\frown$ *f*)
by (*rule OrSChopEqv*)
have *4*: $\vdash$ ((*init w*) $\wedge$ *empty*)$\frown$ *f* = ((*init w*) $\wedge$ *f*)
by (*rule StateAndEmptySChop*)
have *5*: $\vdash$ ($\bigcirc$ (*sfin* (*init w*)))$\frown$ *f* = $\bigcirc$((*sfin* (*init w*))$\frown$ *f*)
by (*rule NextSChop*)
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSChopEqvDiamond*:
$\vdash$ (*sfin* (*init w*))$\frown$ *f* = $\Diamond$ ((*init w*) $\wedge$ *f*)
proof −
have *1*: $\vdash$ (*sfin* (*init w*)) = (#*True*$\frown$((*init w*) $\wedge$ *empty*))
by (*simp add*: *SFinEqvTrueSChopAndEmpty*)
hence *2*: $\vdash$ (*sfin* (*init w*))$\frown$*f* = (#*True*$\frown$((*init w*) $\wedge$ *empty*))$\frown$*f*
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash$ #*True*$\frown$(((*init w*) $\wedge$ *empty*)$\frown$*f*) = (#*True*$\frown$((*init w*) $\wedge$ *empty*))$\frown$*f*
by (*rule SChopAssoc*)
have *4*: $\vdash$ #*True*$\frown$(((*init w*) $\wedge$ *empty*)$\frown$*f*)= $\Diamond$ (((*init w*) $\wedge$ *empty*)$\frown$*f*)
using *TrueSChopEqvDiamond* **by** *blast*
have *5*: $\vdash$ ((*init w*) $\wedge$ *empty*)$\frown$*f* = ((*init w*) $\wedge$ *f*)
using *StateAndEmptySChop* **by** *blast*
hence *6*: $\vdash$ $\Diamond$ (((*init w*) $\wedge$ *empty*)$\frown$*f*) = $\Diamond$ ((*init w*) $\wedge$ *f*)

by (*rule DiamondEqvDiamond*)
from *2 3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSYields*:
$\vdash$ (*sfin* (*init w*)) *syields* (*init w*)
proof −
have *1*: $\vdash$ (*sfin* (*init w*))⌢ (¬(*init w*)) = ◇((*init w*) ∧ ¬(*init w*))
by (*rule SFinSChopEqvDiamond*)
have *2*: $\vdash$ ¬(◇((*init w*) ∧ ¬ (*init w*)))
by (*rule NotDiamondAndNot*)
have *3*: $\vdash$ ¬ ((*sfin* (*init w*))⌢ (¬ (*init w*)))
using *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SFinEQvFinAndFinite*:
$\vdash$ (*finite* ∧ *fin f*) = *sfin f*
by (*metis AndFinEqvChopAndEmpty DiamondSChopdef SFinEqvTrueSChopAndEmpty int-simps*(*20*) *inteq-reflection sometimes-d-def*)

lemma *AndFiniteImpAndSFinStateOrSFinNotState*:
$\vdash$ *f* ∧ *finite* ⟶ (*f* ∧ *sfin* (*init w*)) ∨ (*f* ∧ *sfin* (¬ (*init w*)))
proof −
have *1*: $\vdash$ (¬ *fin* (*init* (¬ *w*)) ∧ *finite*) = (*fin* (*init* (¬ ¬ *w*)) ∧ *finite*)
using *FinNotStateEqvNotFinState* **by** *blast*
have *2*: $\vdash$ (*fin* (*init* (¬ ¬ *w*)) ∧ *finite*) = (*fin* (*init* (*w*)) ∧ *finite*)
by *simp*
have *3*: $\vdash$ *f* ∧ *finite* ⟶ (*f* ∧ *finite*) ∧ *fin* (*init w*) ∨ (*f* ∧ *finite*) ∧ *fin* (¬ *init w*)
by (*simp add*: *ImpAndFinStateOrFinNotState*)
have *4*: $\vdash$ (*finite* ∧ *fin* (*init w*)) = *sfin*(*init w*)
using *SFinEQvFinAndFinite*[*of LIFT*(*init w*)] **by** *fastforce*
have *5*: $\vdash$ ((*f* ∧ *finite*) ∧ *fin* (*init w*)) = (*f* ∧ *sfin* (*init w*))
using *4* **by** *auto*
have *6*: $\vdash$ (*finite* ∧ *fin* (¬(*init w*))) = *sfin*(¬(*init w*))
using *SFinEQvFinAndFinite*[*of LIFT*(¬(*init w*))] **by** *fastforce*
have *7*: $\vdash$ ((*f* ∧ *finite*) ∧ *fin* (¬ (*init w*))) = (*f* ∧ *sfin* (¬ (*init w*)))
using *6* **by** *auto*
show *?thesis*
using *3 5 7* **by** *fastforce*
qed

lemma *AndSFinSChopEqvStateAndSChop*:
$\vdash$ (*f* ∧ *sfin* (*init w*))⌢ *g* = *f*⌢ ((*init w*) ∧ *g*)
proof −
have *1*: $\vdash$ (*sfin* (*init w*)) *syields* (*init w*)
by (*rule SFinSYields*)
have *2*: $\vdash$ *f* ∧ *sfin* (*init w*) ⟶ *sfin* (*init w*)
by *auto*
hence *3*: $\vdash$ (*sfin* (*init w*)) *syields* (*init w*) ⟶

$(f \wedge \ sfin \ (init\ w))\ syields \ (init\ w)$
using *LeftSYieldsImpSYields* **by** *metis*
have *4*: $\vdash (f \wedge \ sfin \ (init\ w))\ syields \ (init\ w)$
using *1 3 MP* **by** *fastforce*
have *5*: $\vdash (f \wedge sfin \ (init\ w))^\frown\ g \wedge (f \wedge \ sfin \ (init\ w))\ syields \ (init\ w)$
$\longrightarrow (f \wedge \ sfin \ (init\ w))^\frown\ (g \wedge (init\ w))$
by (*rule SChopAndSYieldsImp*)
have *6*: $\vdash (f \wedge \ sfin \ (init\ w))^\frown\ g \longrightarrow (f \wedge \ sfin \ (init\ w))^\frown\ (g \wedge (init\ w))$
using *4 5* **by** *fastforce*
have *7*: $\vdash (f \wedge \ sfin \ (init\ w))^\frown\ (g \wedge (init\ w)) \longrightarrow f^\frown\ (g \wedge (init\ w))$
by (*rule AndSChopA*)
have *8*: $\vdash g \wedge (init\ w) \longrightarrow (init\ w) \wedge g$
by *auto*
hence *9*: $\vdash f^\frown\ (g \wedge (init\ w)) \longrightarrow f\ ^\frown((init\ w) \wedge g)$
by (*rule RightSChopImpSChop*)
have *10*: $\vdash (f \wedge \ sfin \ (init\ w))^\frown\ g \longrightarrow f^\frown\ ((init\ w) \wedge g)$
using *6 7 9* **by** *fastforce*
have *11*: $\vdash (f \wedge finite) \longrightarrow (f \wedge \ sfin \ (init\ w)) \vee \ (f \wedge sfin\ (\neg \ (init\ w)))$
using *AndFiniteImpAndSFinStateOrSFinNotState* **by** *blast*
hence *12*: $\vdash f^\frown\ ((init\ w) \wedge g) \longrightarrow$
$((f \wedge \ sfin \ (init\ w)) \vee \ (f \wedge sfin\ (\neg \ (init\ w)))\ )^\frown\ ((init\ w) \wedge g)$
by (*metis FiniteImp LeftChopImpChop inteq-reflection schop-d-def*)
have *13*: $\vdash ((f \wedge \ sfin \ (init\ w)) \vee \ (f \wedge sfin\ (\neg \ (init\ w))))^\frown\ ((init\ w) \wedge g)$
$=$
$((f \wedge sfin\ (init\ w))^\frown\ ((init\ w) \wedge g) \vee \ (f \wedge sfin\ (\neg \ (init\ w)))^\frown((init\ w) \wedge g))$
by (*rule OrSChopEqv*)
have *14*: $\vdash (\ f \wedge sfin \ \ (init\ (\neg\ w)))^\frown\ ((init\ w) \wedge g) \longrightarrow \Diamond(\ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g))$
using *SFinSChopEqvDiamond*
by (*metis SChopImpSChop Prop12 int-iffD1 inteq-reflection lift-and-com*)
have *141*: $\vdash \neg(\ \Diamond(\ \ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g))) \longrightarrow$
$\neg\ (\ (\ f \wedge sfin \ \ \ (init\ (\neg\ w)))^\frown\ ((init\ w) \wedge g))$
using *14* **by** *fastforce*
have *150*: $\vdash ((init\ (\neg\ w)) \wedge ((init\ w) \wedge g)) = \#False$
using *Initprop(2)* **by** *fastforce*
have *15*: $\vdash \neg(\ \Diamond(\ \ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g)))$
by (*metis 150 NotDiamondAndNot int-eq int-simps(21)*)
have *151*: $\vdash \neg\ (\ (\ f \wedge sfin \ \ \ (init\ (\neg\ w)))^\frown\ ((init\ w) \wedge g))$
using *15 141* **by** *fastforce*
have *1511*: $\vdash (\ f \wedge sfin\ (\neg \ \ (init\ w)))^\frown\ ((init\ w) \wedge g) \longrightarrow \#False$
using *151* **by** (*metis Initprop(2) int-simps(14) inteq-reflection*)
have *152*: $\vdash (f \wedge sfin(init\ w))^\frown\ ((init\ w) \wedge g) \vee \ (f \wedge sfin\ (\neg(init\ w)))^\frown((init\ w) \wedge g) \longrightarrow$
$(f \wedge \ sfin \ (init\ w))^\frown\ ((init\ w) \wedge g)$
using *1511* **by** *fastforce*
have *16*: $\vdash f^\frown\ ((init\ w) \wedge g) \longrightarrow (f \wedge \ sfin \ (init\ w))^\frown\ ((init\ w) \wedge g)$
using *12 13 152* **by** *fastforce*
have *17*: $\vdash (f\wedge \ sfin \ (init\ w))^\frown\ ((init\ w) \wedge g) \longrightarrow (f\wedge \ sfin \ (init\ w))^\frown\ g$
by (*rule SChopAndB*)
have *18*: $\vdash f^\frown\ ((init\ w) \wedge g) \longrightarrow (f \wedge \ sfin \ (init\ w))^\frown\ g$
using *16 17* **by** *fastforce*
from *10 18* **show** *?thesis* **by** *fastforce*

qed

lemma *DfAndSFinEqvSChopState*:
$\vdash \quad df\ (f \wedge \ sfin\ \ (init\ w)) = f \frown (init\ w)$
proof −
have *1*: $\vdash (f \wedge sfin(init\ w)) \frown \#True = f \frown ((init\ w) \wedge \#True)$
by (*rule AndSFinSChopEqvStateAndSChop*)
have *2*: $\vdash ((init\ w) \wedge \#True) = (init\ w)$
by *auto*
hence *3*: $\vdash (f \frown ((init\ w) \wedge \#True)) = (f \frown (init\ w))$
by (*rule RightSChopEqvSChop*)
have *4*: $\vdash (f \wedge \ sfin\ \ (init\ w)) \frown \#True = f \frown (init\ w)$
using *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *SFinNotStateEqvNotSFinState*:
$\vdash finite \longrightarrow \ (\neg(\ sfin\ \ (init\ w))\ ) = (sfin\ \ (init\ (\neg\ w))\ \ )$
using *SFinEqvTrueSChopAndEmpty*
by (*metis Initprop*(*2*) *SFinprop*(*3*) *int-eq*)

lemma *BfImpSFinEqvSYieldsState*:
$\vdash \quad bf\ (f \longrightarrow \ sfin\ \ (init\ w)) = f \ \ syields\ \ (init\ w)$
proof −
have *1*: $\vdash \ df\ (f \wedge sfin\ (init\ (\neg\ w))) = f \frown \ \ (init\ (\neg\ w))$
by (*rule DfAndSFinEqvSChopState*)
have *2*: $\vdash finite \longrightarrow (f \wedge sfin(init\ (\neg\ w))) \ = (f \wedge \neg(sfin(init\ w)))$
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
have *3*: $\vdash (f \wedge \neg\ (sfin(init\ w))) = (\neg\ (f \longrightarrow \ sfin\ \ (init\ w)))$
by *auto*
have *4*: $\vdash finite \longrightarrow (f \wedge sfin(init\ \ (\neg\ w))) = (\neg\ (f \longrightarrow \ sfin(init\ w)))$
using *2 3* **by** *fastforce*
hence *5*: $\vdash \ df\ (f \wedge \ sfin\ \ (init\ (\neg\ w))) = \ df\ (\neg\ (f \longrightarrow sfin(init\ w)))$
by (*metis DfEqvNotBfNot FiniteImpAnd df-d-def inteq-reflection schop-d-def*)
have *6*: $\vdash \ df\ (\neg\ (f \longrightarrow \ sfin\ \ (init\ w))) = (\neg(\ bf\ (f \longrightarrow sfin(init\ w))))$
by (*rule DfNotEqvNotBf*)
have *7*: $\vdash \neg\ (bf\ (f \longrightarrow \ sfin\ \ (init\ w))) = f \frown (init\ (\neg\ w))$
using *1 5 6 Initprop* **by** *fastforce*
hence *8*: $\vdash bf\ (f \longrightarrow \ sfin\ \ (init\ w)) = (\neg\ (f \frown (\neg\ \ (init\ w))))$
by (*metis Initprop*(*2*) *int-eq int-simps*(*7*))
from *8* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *StateImpSYields*:
assumes $\vdash \ \ (init\ w) \wedge f \longrightarrow \ sfin\ \ (init\ w1)$
shows $\vdash (init\ w) \longrightarrow (f\ syields\ \ (init\ w1))$
proof −
have *1*: $\vdash (init\ w) \wedge f \ \longrightarrow \ sfin\ \ (init\ w1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash (init\ w) \longrightarrow (f \ \longrightarrow \ sfin\ \ (init\ w1))$ **by** *auto*
hence *3*: $\vdash (init\ w) \longrightarrow bf\ (f \ \longrightarrow \ sfin\ \ (init\ w1))$ **using** *StateImpBfGen* **by** *auto*

have *4*: $\vdash$ *bf* (*f* $\longrightarrow$ *sfin* (*init w1*)) = *f syields* (*init w1*) **by** (*rule BfImpSFinEqvSYieldsState*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndSYieldsImpSYields*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1*
shows $\vdash$ (*init w*) $\wedge$ (*f1 syields g*) $\longrightarrow$ (*f syields g*)
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) $\wedge$ (*f*$\frown$ ($\neg$ *g*)) $\longrightarrow$ *f1*$\frown$ ($\neg$ *g*) **by** (*rule StateAndSChopImpSChopRule*)
hence *3*: $\vdash$ (*init w*) $\wedge$ $\neg$ (*f1*$\frown$ ($\neg$ *g*)) $\longrightarrow$ $\neg$ (*f*$\frown$ ($\neg$ *g*)) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *AndSYieldsA*:
$\vdash$ *f syields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *syields g*
proof −
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftSYieldsImpSYields*)
qed

lemma *AndSYieldsB*:
$\vdash$ *f1 syields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *syields g*
proof −
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f1* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftSYieldsImpSYields*)
qed

lemma *RightSYieldsImpSYields*:
assumes $\vdash$ *g*$\longrightarrow$ *g1*
shows $\vdash$ (*f syields g*) $\longrightarrow$ (*f syields g1*)
proof −
have *1*: $\vdash$ *g* $\longrightarrow$ *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ $\neg$ *g1* $\longrightarrow$ $\neg$ *g* **by** *auto*
hence *3*: $\vdash$ *f*$\frown$ ($\neg$ *g1*) $\longrightarrow$ *f*$\frown$ ($\neg$ *g*) **by** (*rule RightSChopImpSChop*)
hence *4*: $\vdash$ $\neg$ (*f*$\frown$ ($\neg$ *g*)) $\longrightarrow$ $\neg$ (*f*$\frown$ ($\neg$ *g1*)) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *RightSYieldsEqvSYields*:
assumes $\vdash$ *g* = *g1*
shows $\vdash$ (*f syields g*) = (*f syields g1*)
proof −
have *1*: $\vdash$ *g* = *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ ($\neg$ *g*) = ($\neg$ *g1*) **by** *auto*
hence *3*: $\vdash$ *f*$\frown$ ($\neg$ *g*) = *f*$\frown$ ($\neg$ *g1*) **by** (*rule RightSChopEqvSChop*)
hence *4*: $\vdash$ ($\neg$ (*f*$\frown$ ($\neg$ *g*))) = ($\neg$ (*f*$\frown$ ($\neg$ *g1*))) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *BoxImpSYields*:
$\vdash \quad \Box\ g \longrightarrow f\ \mathit{syields}\ \ g$
proof −
have *1*: $\vdash f \frown (\neg\ \ g) \longrightarrow \Diamond(\neg\ \ g)$ **by** (*rule SChopImpDiamond*)
hence *2*: $\vdash \neg\ (\Diamond(\neg\ \ g)) \longrightarrow \neg\ (f \frown (\neg\ \ g))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def always-d-def*)
qed

lemma *BoxEqvTrueSYields*:
$\vdash \quad \Box\ f = \#\mathit{True}\ \mathit{syields}\ \ f$
proof −
have *1*: $\vdash \#\mathit{True} \frown (\neg\ \ f) = \Diamond\ (\neg\ \ f)$ **by** (*rule TrueSChopEqvDiamond*)
hence *2*: $\vdash (\neg\ (\#\mathit{True} \frown (\neg\ \ f))) = (\neg(\ \Diamond\ (\neg\ \ f)))$ **by** *auto*
have *3*: $\vdash \Box\ f = (\neg\ (\ \Diamond\ (\neg\ \ f)))$ **by** (*simp add*: *always-d-def*)
have *4*: $\vdash \Box\ f = (\neg\ (\#\mathit{True} \frown (\neg\ \ f)))$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SYieldsGen*:
assumes $\vdash \quad g$
shows $\quad \vdash f\ \mathit{syields}\ \ g$
proof −
have *1*: $\vdash g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ g$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box\ g \longrightarrow f\ \mathit{syields}\ \ g$ **by** (*rule BoxImpSYields*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SYieldsAndSYieldsEqvSYieldsAnd*:
$\vdash \quad ((f\ \mathit{syields}\ \ g) \wedge (f\ \ \mathit{syields}\ \ g1)) = f\ \mathit{syields}\ (g \wedge g1)$
proof −
have *1*: $\vdash f \frown (\neg\ \ g \vee\ \neg\ \ g1) = ((f \frown (\neg\ \ g)) \vee\ \ (f \frown (\neg\ \ g1)))$ **by** (*rule SChopOrEqv*)
hence *2*: $\vdash ((f \frown (\neg\ \ g)) \vee\ \ (f \frown (\neg\ \ g1))) = f \frown (\neg\ \ g \vee\ \neg\ \ g1)$ **by** *auto*
have *3*: $\vdash (\neg\ \ g \vee\ \neg\ \ g1)\ =\ (\neg\ (g \wedge g1))$ **by** *auto*
hence *4*: $\vdash f \frown (\neg\ \ g \vee\ \neg\ \ g1)\ =\ f \frown (\neg\ (g \wedge g1))$ **by** (*rule RightSChopEqvSChop*)
have *5*: $\vdash (f \frown (\neg\ \ g)) \vee\ \ (f \frown (\neg\ \ g1)) = f \frown (\neg\ (g \wedge g1))$ **using** *2 4* **by** *fastforce*
hence *6*: $\vdash (\neg\ (f \frown (\neg\ \ g)) \wedge \neg\ (f \frown (\neg\ \ g1))) = (\neg\ (f \frown (\neg\ (g \wedge g1))))$ **using** *1 4* **by** *fastforce*
from *6* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SYieldsAndSYieldsImpAndSYieldsAnd*:
$\vdash \quad (f\ \mathit{syields}\ \ g) \wedge (f1\ \mathit{syields}\ \ g1) \longrightarrow (f \wedge f1)\ \mathit{syields}\ (g \wedge g1)$
proof −
have *1*: $\vdash f\ \mathit{syields}\ \ g \longrightarrow (f \wedge f1)\ \mathit{syields}\ \ g$
by (*rule AndSYieldsA*)
have *2*: $\vdash f1\ \mathit{syields}\ \ g1 \longrightarrow (f \wedge f1)\ \mathit{syields}\ \ g1$
by (*rule AndSYieldsB*)
have *3*: $\vdash ((f \wedge f1)\ \mathit{syields}\ \ g \wedge (f \wedge f1)\ \mathit{syields}\ \ g1) = (f \wedge f1)\ \mathit{syields}\ \ (g \wedge g1)$
by (*rule SYieldsAndSYieldsEqvSYieldsAnd*)
from *1 2 3* **show** *?thesis* **by** *fastforce*

qed

lemma *SYieldsSYieldsEqvSChopSYields*:
$\vdash$ *f syields* (*g syields h*) = (*f* ⌢ *g*) *syields h*
proof −
have *1*: $\vdash$ *f* ⌢ (*g* ⌢ (¬ *h*)) = (*f* ⌢ *g*) ⌢ (¬ *h*) **by** (*rule SChopAssoc*)
hence *2*: $\vdash$ *f* ⌢ (*g* ⌢ (¬ *h*)) = (*f* ⌢ *g*) ⌢ (¬ *h*) **by** *auto*
have *3*: $\vdash$ *g* ⌢ (¬ *h*) = (¬ ¬ (*g* ⌢ (¬ *h*))) **by** *auto*
hence *4*: $\vdash$ *f* ⌢ (*g* ⌢ (¬ *h*)) = *f* ⌢ (¬ ¬ (*g* ⌢ (¬ *h*))) **by** (*rule RightSChopEqvSChop*)
have *5*: $\vdash$ *f* ⌢ (¬ ¬ (*g* ⌢ (¬ *h*))) = (*f* ⌢ *g*) ⌢ (¬ *h*) **using** *2 4* **by** *auto*
hence *6*: $\vdash$ *f* ⌢ (¬ (*g syields h*)) = (*f* ⌢ *g*) ⌢ (¬ *h*) **by** (*simp add*: *syields-d-def*)
hence *7*: $\vdash$ (¬ (*f* ⌢ (¬ (*g syields h*)))) = (¬ ((*f* ⌢ *g*) ⌢ (¬ *h*))) **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *EmptySYields*:
$\vdash$ *empty syields f* = *f*
proof −
have *1*: $\vdash$ *empty* ⌢ (¬ *f*) = (¬ *f*) **by** (*rule EmptySChop*)
hence *2*: $\vdash$ (¬ (*empty* ⌢ (¬ *f*))) = *f* **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *NextSYields*:
$\vdash$ (○ *f*) *syields g* = *wnext* (*f syields g*)
proof −
have *1*: $\vdash$ (○ *f*) ⌢ (¬ *g*) = ○(*f* ⌢ (¬ *g*)) **by** (*rule NextSChop*)
hence *2*: $\vdash$ (¬ ((○ *f*) ⌢ (¬ *g*))) = (¬ (○(*f* ⌢ (¬ *g*)))) **by** *auto*
hence *3*: $\vdash$ (○ *f*) *syields g* = (¬ (○(*f* ⌢ (¬ *g*)))) **by** (*simp add*: *syields-d-def*)
have *4*: $\vdash$ (¬(○(*f* ⌢ (¬ *g*)))) = *wnext* (¬ (*f* ⌢ (¬ *g*))) **by** (*auto simp*: *wnext-d-def*)
have *5*: $\vdash$ (○ *f*) *syields g* = *wnext* (¬ (*f* ⌢ (¬ *g*))) **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SkipSChopEqvNext*:
$\vdash$ *skip* ⌢ *f* = ○ *f*
by (*meson NextSChopdef Prop11*)

lemma *SkipSYieldsEqvWeakNext*:
$\vdash$ *skip syields f* = *wnext f*
proof −
have *1*: $\vdash$ *skip* ⌢ (¬ *f*) = ○(¬ *f*) **by** (*rule SkipSChopEqvNext*)
hence *2*: $\vdash$ (¬ (*skip* ⌢ (¬ *f*))) = (¬(○(¬ *f*))) **by** *auto*
have *3*: $\vdash$ (¬ (○(¬ *f*))) = *wnext f* **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash$ (¬ (*skip* ⌢ (¬ *f*))) = *wnext f* **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *NextImpSkipSYields*:
$\vdash$ ○ *f* ⟶ *skip syields f*

proof −
have *1*: ⊢ ○ *f* ⟶ *wnext f* **using** *WnextEqvEmptyOrNext* **by** *fastforce*
have *2*: ⊢ *skip syields f* = *wnext f* **by** (*rule SkipSYieldsEqvWeakNext*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreEqvSkipSChopTrue*:
⊢ *more* = *skip* ⌢ #*True*
proof −
have *1*: ⊢ *skip* ⌢ #*True* = ○#*True* **by** (*rule SkipSChopEqvNext*)
hence *2*: ⊢ ○#*True* = *skip* ⌢ #*True* **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *more-d-def*)
qed

lemma *MoreSChopImpMore*:
⊢ *more* ⌢ *f* ⟶ *more*
proof −
have *1*: ⊢ (○#*True*)⌢ *f* = ○(#*True*⌢ *f*)
by (*rule NextSChop*)
have *2*: ⊢ ○(#*True*⌢ *f*) ⟶ *more*
by (*metis DiIntro LeftChopImpMoreRule di-d-def more-d-def next-d-def*)
have *3*: ⊢ (○#*True*⌢ *f*) ⟶ *more*
using *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*metis more-d-def*)
qed

lemma *MoreSChopImpFmore*:
⊢ *more* ⌢ (*f* ∧ *finite*) ⟶ *fmore*
proof −
have *1*: ⊢ *more* ⌢ (*f* ∧ *finite*)= ○(#*True*⌢ (*f* ∧ *finite*))
by (*simp add*: *NextSChop more-d-def*)
have *2*: ⊢ ○(#*True*⌢ (*f* ∧ *finite*)) ⟶ *fmore*
by (*metis 1 FmoreChopImpFmore fmore-d-def int-eq schop-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopMoreImpMore*:
⊢ *f*⌢ *more* ⟶ *more*
proof −
have *1*: ⊢ *f* ⌢ *more* ⟶ ◇ *more* **by** (*rule SChopImpDiamond*)
have *2*: ⊢ ◇ *more* ⟶ *more* **by** (*simp add*: *FiniteChopMoreEqvMore int-iffD1 sometimes-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreSChopEqvNextDiamond*:
⊢ *more* ⌢ *f* = ○(◇ *f*)
proof −
have *1*: ⊢ *more* ⌢ *f* = (○ #*True*)⌢ *f* **by** (*simp add*: *more-d-def*)
have *2*: ⊢ (○ #*True*)⌢ *f* = ○(#*True*⌢ *f*) **by** (*rule NextSChop*)
have *3*: ⊢ *more* ⌢ *f* = ○(#*True*⌢ *f*) **using** *1 2* **by** *fastforce*

from *3* **show** *?thesis* **by** (*metis TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *WeakNextBoxImpMoreSYields*:
$\vdash$ *more syields f* = *wnext*($\Box$ *f*)
proof −
have *1*: $\vdash$ *more* $\frown$ ($\neg$ *f*) = $\bigcirc$($\Diamond$ ($\neg$*f*)) **by** (*rule MoreSChopEqvNextDiamond*)
have *2*: $\vdash$ $\bigcirc$($\Diamond$ ($\neg$*f*)) = $\bigcirc$($\neg$($\Box$*f*)) **by** (*auto simp*: *always-d-def*)
have *3*: $\vdash$ $\bigcirc$($\neg$($\Box$*f*)) = ($\neg$ (*wnext*($\Box$ *f*))) **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash$ *more* $\frown$ ($\neg$ *f*) = ($\neg$(*more syields f*)) **by** (*simp add*: *syields-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotEqvSYieldsMore*:
$\vdash$ *finite* $\longrightarrow$ ($\neg$ *f*) = *f syields more*
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ *f*$\frown$ *empty* = *f* **by** (*rule SChopEmpty*)
hence *2*: $\vdash$ *finite* $\longrightarrow$ ($\neg$ (*f*$\frown$ *empty*)) = ($\neg$ *f*) **by** *auto*
have *3*: $\vdash$ *empty* = ($\neg$ *more*) **by** (*auto simp*: *empty-d-def*)
hence *4*: $\vdash$ *f*$\frown$ *empty* = *f*$\frown$ ($\neg$ *more*) **by** (*rule RightSChopEqvSChop*)
hence *5*: $\vdash$ ($\neg$ (*f*$\frown$ *empty*)) = ($\neg$ (*f*$\frown$ ($\neg$ *more*))) **by** *auto*
have *6*: $\vdash$ *finite* $\longrightarrow$ ($\neg$ *f*) = ($\neg$ (*f*$\frown$ ($\neg$ *more*))) **using** *2 5* **by** *fastforce*
from *6* **show** *?thesis* **by** (*metis syields-d-def*)
qed

lemma *LeftSChopImpMoreRule*:
assumes $\vdash$ *f* $\longrightarrow$ *more*
shows $\vdash$ *f*$\frown$ *g* $\longrightarrow$ *more*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *more* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*$\frown$ *g* $\longrightarrow$ *more* $\frown$ *g* **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash$ *more* $\frown$ *g* $\longrightarrow$ *more* **by** (*rule MoreSChopImpMore*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *LeftSChopImpFMoreRule*:
assumes $\vdash$ *f* $\longrightarrow$ *fmore*
shows $\vdash$ *f*$\frown$ (*g* $\wedge$ *finite*) $\longrightarrow$ *fmore*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *fmore*
using *assms* **by** *auto*
hence *2*: $\vdash$ *f*$\frown$ (*g* $\wedge$ *finite*) $\longrightarrow$ *more* $\frown$ (*g* $\wedge$ *finite*)
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
FmoreEqvSkipChopFinite LeftSChopImpSChop Prop12 inteq-reflection)
have *3*: $\vdash$ *more* $\frown$ (*g* $\wedge$ *finite*) $\longrightarrow$ *fmore*
using *MoreSChopImpFmore* **by** *fastforce*
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *RightSChopImpMoreRule*:

assumes $\vdash \quad g \longrightarrow \ more$
shows $\quad \vdash f \frown g \longrightarrow \ more$
proof −
have *1*: $\vdash g \longrightarrow \ more$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown \ more$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown \ more \longrightarrow \ more$ **by** (*rule SChopMoreImpMore*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NotDfEqvBfNot*:
$\vdash \quad (\neg \ (\ df \ \ f)) = bf \ (\neg \ \ f)$
proof −
have *1*: $\vdash f = (\neg \neg \ \ f)$ **by** *auto*
hence *2*: $\vdash \ df \ \ f = \ df \ (\neg \neg \ \ f)$ **by** (*rule DfEqvDf*)
hence *3*: $\vdash (\neg \ \ (\ df \ \ f)) = (\neg \ (\ df \ (\neg \neg \ \ f)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

lemma *SChopImpDf*:
$\vdash \quad f \frown g \longrightarrow \ df \ \ f$
proof −
have *1*: $\vdash g \longrightarrow \#True$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown \#True$ **by** (*rule RightSChopImpSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *TrueEqvTrueSChopTrue*:
$\vdash \quad \#True = \#True \frown \#True$
proof −
have *1*: $\vdash \#True \frown \#True \longrightarrow \#True$
by *auto*
have *2*: $\vdash \#True \longrightarrow \#True \frown \#True$
by (*metis DfState Initprop*(*4*) *df-d-def int-eq-true int-iffD1 inteq-reflection*)
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *DfEqvDfDf*:
$\vdash \quad \ df \ \ f = \ df \ (\ \ df \ \ f)$
proof −
have *1*: $\vdash \#True = \#True \frown \#True$ **by** (*rule TrueEqvTrueSChopTrue*)
hence *2*: $\vdash f \frown \#True = f \frown (\#True \frown \#True)$ **by** (*rule RightSChopEqvSChop*)
have *3*: $\vdash f \frown (\#True \frown \#True) = (f \frown \#True) \frown \#True$ **by** (*rule SChopAssoc*)
have *4*: $\vdash f \frown \#True = (f \frown \#True) \frown \#True$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis df-d-def*)
qed

lemma *BfEqvBfBf*:

$\vdash$ $bf\, f = bf(\ bf\ \ f)$
proof −
have *1*: $\vdash$ $df\ (\neg\ \ f) =\ \ df(\ \ df\ (\neg\ \ f))$ **by** (*rule DfEqvDfDf*)
have *2*: $\vdash$ $df\ (\neg\ \ f) = (\neg\ (\ bf\ \ f))$ **by** (*rule DfNotEqvNotBf*)
hence *3*: $\vdash$ $df\ (\ df\ (\neg\ \ f)) =\ \ df\ (\neg\ (\ bf\ \ f))$ **by** (*rule DfEqvDf*)
have *4*: $\vdash$ $df\ (\neg\ \ f) =\ \ df\ (\neg(\ \ bf\ \ f))$ **using** *1 3* **by** *fastforce*
hence *5*: $\vdash (\neg\ (\ df\ (\neg\ \ f))) = (\neg\ (\ df\ (\neg\ (\ bf\ \ f))))$ **by** *fastforce*
from *5* **show** *?thesis* **by** (*metis bf-d-def*)
qed

lemma *BfImpBfBf*:
$\vdash$ $bf\, f \longrightarrow bf(bf\, f)$
proof −
have *1*: $\vdash$ $bf(bf\, f) = bf\, f$ **using** *BfEqvBfBf* **by** *fastforce*
from *1* **show** *?thesis* **by** (*simp add*: *int-iffD2*)
qed

lemma *DfOrEqv*:
$\vdash$ $\ \ df\ (f \vee\ \ g) =\ \ (df\ \ f \vee\ \ \ df\ \ g)$
proof −
have *1*: $\vdash$ $(f \vee\ \ g)^\frown\, \#True = (f^\frown\, \#True \vee\ \ g^\frown\, \#True)$ **by** (*rule OrSChopEqv*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfAndA*:
$\vdash$ $\ \ df\ (f \wedge g) \longrightarrow\ \ df\ \ f$
proof −
have *1*: $\vdash$ $(f \wedge g)^\frown\, \#True \longrightarrow f^\frown\, \#True$ **by** (*rule AndSChopA*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfAndB*:
$\vdash$ $\ \ df\ (f \wedge g) \longrightarrow\ \ df\ \ g$
proof −
have *1*: $\vdash$ $(f \wedge g)^\frown\, \#True \longrightarrow g^\frown\, \#True$ **by** (*rule AndSChopB*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfAndImpAnd*:
$\vdash$ $\ \ df\ (f \wedge g) \longrightarrow\ \ df\ \ f \wedge\ \ df\ \ g$
proof −
have *1*: $\vdash$ $\ \ df\ (f \wedge g) \longrightarrow\ \ df\ \ f$ **by** (*rule DfAndA*)
have *2*: $\vdash$ $\ \ df\ (f \wedge g) \longrightarrow\ \ df\ \ g$ **by** (*rule DfAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DfSkipEqvMore*:
$\vdash$ $\ \ df\ \ skip\ \ =\ \ more$

proof −
have *1*: ⊢ *skip* ⌢ #*True* = ○#*True* **by** (*rule SkipSChopEqvNext*)
have *2*: ⊢ ○#*True* = *more* **by** (*auto simp*: *more-d-def*)
have *3*: ⊢ *skip* ⌢ #*True* = *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfMoreEqvMore*:
⊢ *df more* = *more*
proof −
have *1*: ⊢ *df* (○ #*True*) = ○(*df* #*True*) **by** (*rule DfNext*)
have *2*: ⊢ ○(*df* #*True*) ⟶ *more*
by (*metis ChopImpDi di-d-def more-d-def next-d-def*)
have *3*: ⊢ *df*(○ #*True*) ⟶ *more* **using** *1 2* **by** *fastforce*
hence *4*: ⊢ *df more* ⟶ *more* **by** (*simp add*: *more-d-def*)
have *5*: ⊢ *more* ⟶ *df more*
by (*metis 1 4 TrueEqvTrueSChopTrue df-d-def inteq-reflection more-d-def*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *DfIfEqvRule*:
assumes ⊢ *f* = *if*$_i$ (*init w*) *then g else h*
shows ⊢ *df f* = *if*$_i$ (*init w*) *then* (*df g*) *else* (*df h*)
proof −
have *1*: ⊢ *f* = *if*$_i$ (*init w*) *then g else h* **using** *assms* **by** *auto*
hence *2*: ⊢ *f*⌢ #*True* = *if*$_i$ (*init w*) *then* (*g*⌢ #*True*) *else* (*h*⌢ #*True*) **by** (*rule IfSChopEqvRule*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *SDaNotEqvNotSBa*:
⊢ *sda* (¬ *f*) = (¬ (*sba f*))
proof −
have *1*: ⊢ *sba f* = (¬ (*sda* (¬ *f*))) **by** (*simp add*: *sba-d-def*)
from *1* **show** *?thesis* **by** *fastforce*
qed

lemma *SDaEqvSDa*:
assumes ⊢ *f* = *g*
shows ⊢ *sda f* = *sda g*
using *assms* **using** *int-eq* **by** *force*

lemma *SDaEqvNotSBaNot*:
⊢ *sda f* = (¬ (*sba* (¬ *f*)))
proof −
have *1*: ⊢ *sba* (¬ *f*) = (¬ (*sda* (¬ ¬ *f*))) **by** (*simp add*: *sba-d-def*)
hence *2*: ⊢ *sda* (¬ ¬ *f*) = (¬(*sba* (¬ *f*))) **by** *fastforce*
have *3*: ⊢ *f* = (¬ ¬ *f*) **by** *simp*
hence *4*: ⊢ *sda f* = *sda* (¬ ¬ *f*) **by** (*rule SDaEqvSDa*)
from *2 4* **show** *?thesis* **by** *simp*
qed

lemma *SBaElim*:
$\vdash \quad sba\ f \wedge finite \longrightarrow f$
proof −
have *1*: $\vdash\ sba\ \ f = \Box(bf\ \ f)$
by (*rule SBaEqvBtBf*)
have *2*: $\vdash\ bf\ \ f \wedge finite \longrightarrow f$
by (*rule BfElim*)
hence *3*: $\vdash \Box(bf\ \ f \wedge finite \longrightarrow f)$
by (*rule BoxGen*)
have *4*: $\vdash \Box(bf\ \ f\ \wedge finite \longrightarrow f) \longrightarrow \Box(bf\ \ f \wedge finite) \longrightarrow \Box\ f$
by (*rule BoxImpDist*)
have *5*: $\vdash \Box(bf\ \ f \wedge finite) \longrightarrow \Box\ f$
using *3 4 MP* **by** *fastforce*
have *6*: $\vdash \Box(bf\ \ f \wedge finite) = (\Box(bf\ \ f) \wedge finite)$
by (*metis* (*no-types*, *lifting*) *BoxEqvFiniteYields FiniteChopInfEqvInf NotChopEqvYieldsNot YieldsAndYieldsEqvYieldsAnd finite-d-def inteq-reflection*)
have *7*: $\vdash \Box\ f \longrightarrow f$
by (*rule BoxElim*)
from *1 5 6 7* **show** *?thesis* **using** *SBaImpBt lift-imp-trans* **by** *metis*
qed

lemma *SDaIntro*:
$\vdash\ \ f \wedge finite \longrightarrow\ \ sda\ \ f$
proof −
have *1*: $\vdash\ \ sba\ \ (\neg\ \ f) \wedge finite \longrightarrow (\neg\ \ f)$ **by** (*rule SBaElim*)
hence *2*: $\vdash \neg\ \neg\ \ f \longrightarrow \neg\ (\ \ sba\ \ (\neg\ \ f) \wedge finite)$ **by** *fastforce*
have *3*: $\vdash f = (\neg\ \neg\ \ f)$ **by** *simp*
have *4*: $\vdash\ \ sda\ \ f = (\neg\ \ (\ sba\ \ (\neg\ \ f)))$ **by** (*rule SDaEqvNotSBaNot*)
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaGen*:
assumes $\vdash\ \ f$
shows $\vdash\ \ sba\ \ f$
proof −
have *1*: $\vdash\ \ \ f$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ f$ **by** (*rule BoxGen*)
hence *3*: $\vdash bf(\ \Box\ f)$ **by** (*rule BfGen*)
have *4*: $\vdash\ \ sba\ \ f = bf\ (\Box\ f)$ **by** (*rule SBaEqvBfBt*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpDist*:
$\vdash\ \ sba\ (f \longrightarrow g) \longrightarrow\ \ sba\ \ f \longrightarrow\ \ sba\ \ g$
proof −
have *1*: $\vdash bf\ (f \longrightarrow g) \longrightarrow (bf\ \ f \longrightarrow bf\ \ g)$
by (*rule BfImpDist*)
hence *2*: $\vdash \Box(bf\ (f \longrightarrow g) \longrightarrow (bf\ \ f \longrightarrow bf\ \ g))$
by (*rule BoxGen*)

have *3*: $\vdash \Box(bf\ (f \longrightarrow g) \longrightarrow (bf\ \ f \longrightarrow bf\ \ g))$
$\longrightarrow$
$(\Box\ (bf\ (f \longrightarrow g)) \longrightarrow (\Box(bf\ \ f) \longrightarrow \Box(bf\ \ g)))$
by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)
have *4*: $\vdash \Box(bf\ (f \longrightarrow g)) \longrightarrow (\Box(bf\ \ f) \longrightarrow \Box(bf\ \ g))$
using *2 3 MP* **by** *fastforce*
have *5*: $\vdash\ \ sba\ (f \longrightarrow g) = \Box(bf\ (f \longrightarrow g))$
by (*rule SBaEqvBtBf*)
have *6*: $\vdash\ \ sba\ \ f = \Box(bf\ \ f)$
by (*rule SBaEqvBtBf*)
have *7*: $\vdash\ \ sba\ \ g = \Box(bf\ \ g)$
by (*rule SBaEqvBtBf*)
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaAndEqv*:
$\vdash\ \ \ sba\ (f \wedge g) =\ (sba\ \ f \wedge\ \ sba\ \ g)$
proof −
have *1*: $\vdash\ \ \ sba\ (f \wedge g) =\ \ \Box(bf\ (f \wedge g))$
by (*rule SBaEqvBtBf*)
have *2*: $\vdash\ \ bf\ (f \wedge g) = (bf\ f \wedge bf\ g)$
by (*simp add*: *BfAndEqvBfAndBf*)
hence *3*: $\vdash \Box(bf\ (f \wedge g)) = \Box(bf\ f \wedge bf\ g)$
using *BoxEqvBox* **by** *blast*
have *4*: $\vdash \Box(bf\ f \wedge bf\ g) = (\Box(bf\ f)\ \ \wedge \Box(bf\ g))$
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have *5*: $\vdash\ sba\ \ f = \Box(bf\ \ f)$
by (*rule SBaEqvBtBf*)
have *6*: $\vdash\ \ sba\ \ g = \Box(bf\ \ g)$
by (*rule SBaEqvBtBf*)
from *1 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpSBaEqvSBa*:
$\vdash\ \ sba\ (f = g) \longrightarrow\ (sba\ \ f =\ \ sba\ \ g)$
proof −
have *1*: $\vdash\ \ sba\ (f \longrightarrow g) \longrightarrow\ \ sba\ \ f \longrightarrow\ \ sba\ \ g$
by (*rule SBaImpDist*)
have *2*: $\vdash\ \ sba\ (g \longrightarrow f) \longrightarrow\ \ sba\ \ g \longrightarrow\ \ sba\ \ f$
by (*rule SBaImpDist*)
have *3*: $\vdash\ \ (f = g) =\ \ ((f \longrightarrow g) \wedge (g \longrightarrow f))$
by *auto*
hence *31*: $\vdash\ sba(f = g) =\ \ sba\ ((f \longrightarrow g) \wedge (g \longrightarrow f))$
using *inteq-reflection* **by** *force*
have *4*: $\vdash\ sba\ ((f \longrightarrow g) \wedge (g \longrightarrow f)) = (sba((f \longrightarrow g)) \wedge sba((g \longrightarrow f)))$
by (*rule SBaAndEqv*)
have *5*: $\vdash ((sba\ \ f \longrightarrow\ \ sba\ \ g) \wedge (sba\ \ g \longrightarrow\ \ sba\ \ f)) = (sba\ \ f =\ \ sba\ \ g)$
by *auto*
from *1 2 31 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpSBa*:
assumes $\vdash \ f \longrightarrow g$
shows $\vdash sba \ f \longrightarrow \ sba \ g$
using *SBaGen SBaImpDist MP assms* **by** *metis*

lemma *SBaEqvSBa*:
assumes $\vdash \ f = g$
shows $\vdash sba \ f = \ sba \ g$
using *SBaGen SBaImpSBaEqvSBa MP assms* **by** *metis*

lemma *SDaImpSDa*:
assumes $\vdash \ f \longrightarrow g$
shows $\vdash sda \ f \longrightarrow \ sda \ g$
using *assms* **by** (*metis SDaEqvDtDf DfAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *SDaEqvSDaSDa*:
$\vdash \ sda \ f = \ sda \ (\ sda \ f)$
proof −
have *1*: $\vdash \ sda \ f = \Diamond(\ df \ f)$
by (*rule SDaEqvDtDf*)
have *2*: $\vdash \ df \ f = \ (df \ (\ df \ f))$
by (*rule DfEqvDfDf*)
hence *3*: $\vdash \Diamond \ (\ df \ f) = \Diamond \ (df \ \ (df \ f))$
by (*rule DiamondEqvDiamond*)
have *4*: $\vdash \Diamond \ \ (df \ f) = \Diamond(\Diamond \ (df \ \ (df \ f)))$
using *DiamondEqvDiamondDiamond DfEqvDfDf* **using** *3* **by** *fastforce*
have *5*: $\vdash \Diamond \ \ (df \ \ (df \ f)) = \ df \ (\Diamond \ \ (df \ f))$
by (*rule DtDfEqvDfDt*)
hence *6*: $\vdash \Diamond(\Diamond \ \ (df \ \ (df \ f))) = \Diamond \ (df \ (\Diamond \ \ (df \ f)))$
by (*rule DiamondEqvDiamond*)
have *7*: $\vdash \ sda \ f = \Diamond \ (df(\ \Diamond \ (\ df \ f)))$
using *1 3 4 6* **by** *fastforce*
have *8*: $\vdash \ sda \ (\Diamond \ (\ df \ f)) = \Diamond(\ df \ (\Diamond \ \ (df \ f)))$
by (*rule SDaEqvDtDf*)
have *9*: $\vdash \ sda \ (\ sda \ f) = \ sda \ (\Diamond \ \ (df \ f))$
using *1* **by** (*rule SDaEqvSDa*)
from *7 8 9* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaEqvSBaSBa*:
$\vdash \ \ sba \ f = \ sba \ \ (sba \ f)$
proof −
have *1*: $\vdash \ sda \ (\neg \ f) = \ sda \ \ (sda \ (\neg \ f))$ **by** (*rule SDaEqvSDaSDa*)
have *2*: $\vdash \ sda \ \ (sda \ (\neg \ f)) = (\neg \ \ (sba \ (\neg \ \ (sda \ (\neg \ f)))))$ **by** (*rule SDaEqvNotSBaNot*)
have *3*: $\vdash (\neg \ \ (sda \ \ (sda \ (\neg \ f)))) = \ sba \ (\neg \ \ (sda \ (\neg \ f)))$ **by** (*auto simp*: *sba-d-def*)
have *4*: $\vdash (\neg \ \ (sda \ (\neg \ f))) = \ sba \ (\neg \ \ (sda \ (\neg \ f)))$ **using** *1 2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis sba-d-def*)
qed

lemma *SBaLeftSChopImpSChop*:
$\vdash$ *sba* (*f*⟶ *f1*) ⟶ *f*⌢ *g* ⟶ *f1*⌢ *g*
proof −
have *1*: $\vdash$ *sba* (*f* ⟶ *f1*) ⟶ *bf* (*f* ⟶ *f1*) **by** (*rule SBaImpBf*)
have *2*: $\vdash$ *bf* (*f* ⟶ *f1*) ⟶ *f*⌢ *g* ⟶ *f1*⌢ *g* **by** (*rule BfSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaLeftSChopImpSChop*:
$\vdash$ *ba* (*f*⟶ *f1*) ⟶ *f*⌢ *g* ⟶ *f1*⌢ *g*
proof −
have *1*: $\vdash$ *ba* (*f* ⟶ *f1*) ⟶ *bf* (*f* ⟶ *f1*) **by** (*rule BaImpBf*)
have *2*: $\vdash$ *bf* (*f* ⟶ *f1*) ⟶ *f*⌢ *g* ⟶ *f1*⌢ *g* **by** (*rule BfSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaRightSChopImpSChop*:
$\vdash$ *sba* (*g*⟶ *g1*) ∧ *finite* ⟶ *f*⌢ *g* ⟶ *f*⌢ *g1*
proof −
have *1*: $\vdash$ *sba* (*g* ⟶ *g1*) ∧ *finite* ⟶ □(*g* ⟶ *g1*) **by** (*rule SBaImpBt*)
have *2*: $\vdash$ □(*g* ⟶ *g1*) ⟶ *f*⌢ *g* ⟶ *f*⌢ *g1* **by** (*rule BoxSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaRightSChopImpSChop*:
$\vdash$ *ba* (*g*⟶ *g1*) ⟶ *f*⌢ *g* ⟶ *f*⌢ *g1*
proof −
have *1*: $\vdash$ *ba* (*g* ⟶ *g1*) ⟶ □(*g* ⟶ *g1*) **by** (*rule BaImpBt*)
have *2*: $\vdash$ □(*g* ⟶ *g1*) ⟶ *f*⌢ *g* ⟶ *f*⌢ *g1* **by** (*rule BoxSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndSBaImport*:
$\vdash$ (*f*⌢ *f1*) ∧ *sba g* ∧ *finite* ⟶ (*f* ∧ *g*)⌢ (*f1* ∧ *g*)
proof −
have *1*: $\vdash$ *sba g* ∧ *finite* ∧ (*f*⌢ *f1*) ⟶ (*g* ∧ *f*)⌢ (*g* ∧ *f1*) **by** (*rule SBaAndSChopImport*)
have *2*: $\vdash$ (*g* ∧ *f*)⌢ (*g* ∧ *f1*) = (*f* ∧ *g*)⌢ (*f1* ∧ *g*) **by** (*rule AndSChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndBaImport*:
$\vdash$ (*f*⌢ *f1*) ∧ *ba g* ⟶ (*f* ∧ *g*)⌢ (*f1* ∧ *g*)
proof −
have *1*: $\vdash$ *ba g* ∧ (*f*⌢ *f1*) ⟶ (*g* ∧ *f*)⌢ (*g* ∧ *f1*) **by** (*rule BaAndSChopImport*)
have *2*: $\vdash$ (*g* ∧ *f*)⌢ (*g* ∧ *f1*) = (*f* ∧ *g*)⌢ (*f1* ∧ *g*) **by** (*rule AndSChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndSChopImportA*:
$\vdash$ *ba f* ∧ *g*⌢*g1* ⟶ (*f* ∧ *g*)⌢*g1*

by (*meson BaAndSChopImport SChopAndB lift-imp-trans*)

lemma *BaAndSChopImportB*:
$\vdash$ *ba f* $\wedge$ *g*$\frown$*g1* $\longrightarrow$ (*f* $\wedge$ *g*)$\frown$(*ba f* $\wedge$ *g1*)
proof −
have *1*: $\vdash$ *ba f* = *ba* (*ba f*)
by (*simp add*: *BaEqvBaBa*)
have *2*: $\vdash$ *ba* (*ba f*) $\wedge$ *g*$\frown$*g1* $\longrightarrow$ *g*$\frown$(*ba f* $\wedge$ *g1*)
by (*metis AndSChopB BaAndSChopImport lift-imp-trans*)
have *3*: $\vdash$ *ba f* $\wedge$ *g*$\frown$(*ba f* $\wedge$ *g1*) $\longrightarrow$ (*f* $\wedge$ *g*)$\frown$(*ba f* $\wedge$ *g1*)
by (*simp add*: *BaAndSChopImportA*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpSBaImpSBaAnd*:
$\vdash$ *sba h* $\longrightarrow$ *sba*(*g* $\longrightarrow$ *sba h* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *sba h* $\longrightarrow$ (*g* $\longrightarrow$ *sba h* $\wedge$ *g*) **by** *fastforce*
hence *2*: $\vdash$ *sba*(*sba h*) $\longrightarrow$ *sba*(*g* $\longrightarrow$ *sba h* $\wedge$ *g*) **by** (*rule SBaImpSBa*)
have *3*: $\vdash$ *sba h* = *sba*(*sba h*) **by** (*rule SBaEqvSBaSBa*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaSChopImpSChopSBa*:
$\vdash$ *sba f* $\wedge$ *finite* $\longrightarrow$ *g*$\frown$ *g1* $\longrightarrow$ *g*$\frown$ ((*sba f*) $\wedge$ *g1*)
proof −
have *1*: $\vdash$ *sba f* $\longrightarrow$ *sba* (*g1* $\longrightarrow$ (*sba f*) $\wedge$ *g1*)
by (*rule SBaImpSBaImpSBaAnd*)
have *2*: $\vdash$ *sba* (*g1* $\longrightarrow$ *sba f* $\wedge$ *g1*) $\wedge$ *finite* $\longrightarrow$ *g*$\frown$ *g1* $\longrightarrow$ *g*$\frown$ (*sba f* $\wedge$ *g1*)
by (*rule SBaRightSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaSChopImpSChopBa*:
$\vdash$ *ba f* $\longrightarrow$ *g*$\frown$ *g1* $\longrightarrow$ *g*$\frown$ ((*ba f*) $\wedge$ *g1*)
proof −
have *1*: $\vdash$ *ba f* $\longrightarrow$ *ba* (*g1* $\longrightarrow$ (*ba f*) $\wedge$ *g1*)
by (*rule BaImpBaImpBaAnd*)
have *2*: $\vdash$ *ba* (*g1* $\longrightarrow$ *ba f* $\wedge$ *g1*) $\longrightarrow$ *g*$\frown$ *g1* $\longrightarrow$ *g*$\frown$ (*ba f* $\wedge$ *g1*)
by (*rule BaRightSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DfNotSBaImpNotSBa*:
$\vdash$ *df* (¬ (*sba f*)) $\longrightarrow$ ¬ (*sba f*)
proof −
have *1*: $\vdash$ *sba f* = *sba*(*sba f*) **by** (*rule SBaEqvSBaSBa*)
have *2*: $\vdash$ *sba* (*sba f*) $\longrightarrow$ *bf* (*sba f*) **by** (*rule SBaImpBf*)
have *3*: $\vdash$ *sba f* $\longrightarrow$ *bf* (*sba f*) **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *sba f* $\longrightarrow$ ¬ (*df* (¬ (*sba f*))) **by** (*simp add*: *bf-d-def*)

from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *DfNotBaImpNotBa*:
$\vdash \ df\ (\neg\ (\ ba\ \ f)) \longrightarrow \neg\ (\ ba\ \ f)$
proof −
have *1*: $\vdash \ ba\ \ f = \ ba(\ \ ba\ \ f)$ **by** (*rule BaEqvBaBa*)
have *2*: $\vdash \ ba\ (\ ba\ \ f) \longrightarrow bf\ (\ ba\ \ f)$ **by** (*rule BaImpBf*)
have *3*: $\vdash \ ba\ \ f \longrightarrow bf\ (\ ba\ \ f)$ **using** *1 2* **by** *fastforce*
hence *4*: $\vdash \ ba\ \ f \longrightarrow \neg\ (\ df\ (\neg\ (\ ba\ \ f)))$ **by** (*simp add*: *bf-d-def*)
from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotSBaSChopImpNotSBa*:
$\vdash \ (\neg\ (\ sba\ \ f))^\frown\ g \longrightarrow \neg\ (\ sba\ \ f)$
proof −
have *1*: $\vdash (\neg\ (\ sba\ \ f))^\frown\ g \longrightarrow \ df\ (\neg\ (\ sba\ \ f))$ **by** (*rule SChopImpDf*)
have *2*: $\vdash \ df\ (\neg\ (\ sba\ \ f)) \longrightarrow \neg\ (\ sba\ \ f)$ **by** (*rule DfNotSBaImpNotSBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NotBaSChopImpNotSBa*:
$\vdash \ (\neg\ (\ ba\ \ f))^\frown\ g \longrightarrow \neg\ (\ ba\ \ f)$
proof −
have *1*: $\vdash (\neg\ (\ ba\ \ f))^\frown\ g \longrightarrow \ df\ (\neg\ (\ ba\ \ f))$ **by** (*rule SChopImpDf*)
have *2*: $\vdash \ df\ (\neg\ (\ ba\ \ f)) \longrightarrow \neg\ (\ ba\ \ f)$ **by** (*rule DfNotBaImpNotBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DiamondSFinImpSFin*:
$\vdash \Diamond\ (sfin\ f) \longrightarrow sfin\ f$
proof −
have *1*: $\vdash sfin\ f = \#True^\frown(f \wedge empty)$
by (*rule SFinEqvTrueSChopAndEmpty*)
hence *2*: $\vdash \Diamond\ (sfin\ f) = \#True^\frown(\#True^\frown(f \wedge empty))$
by (*metis DiamondSChopdef inteq-reflection*)
have *3*: $\vdash \#True^\frown(\#True^\frown(f \wedge empty)) = (\#True^\frown\#True)^\frown(f \wedge empty)$
by (*rule SChopAssoc*)
have *4*: $\vdash (\#True^\frown\#True)^\frown(f \wedge empty) \longrightarrow \#True^\frown(f \wedge empty)$
using *1 2 3*
by (*metis SChopImpDiamond TrueEqvTrueSChopTrue inteq-reflection*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopSFinImpSFin*:
$\vdash \ f^\frown\ sfin\ \ (init\ w) \longrightarrow \ sfin\ \ (init\ w)$
proof −
have *1*: $\vdash f^\frown\ sfin\ \ (init\ w) \longrightarrow \Diamond\ (\ sfin\ \ (init\ w))$ **by** (*rule SChopImpDiamond*)
have *2*: $\vdash \Diamond\ \ (sfin\ \ (init\ w)) \longrightarrow \ sfin\ \ (init\ w)$ **by** (*rule DiamondSFinImpSFin*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*

qed

lemma *SFinImpSYieldsSFin*:
$\vdash$ *sfin* (*init w*) $\longrightarrow$ *f syields* (*sfin* (*init w*))
proof −
have *1*: $\vdash$ *f*$\frown$ (*sfin* (*init* ($\neg$ *w*))) $\longrightarrow$ (*sfin* (*init* ($\neg$ *w*)))
by (*simp add*: *SChopSFinImpSFin*)
have *2*: $\vdash$ *finite* $\longrightarrow$ ($\neg$ (*sfin* (*init w*))) = (*sfin* (*init* ($\neg$ *w*)))
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
hence *3*: $\vdash$ *finite* $\longrightarrow$ *f* $\frown$ ($\neg$ (*sfin* (*init w*))) = *f*$\frown$ (*sfin* (*init* ($\neg$ *w*)))
using *FiniteRightSChopEqvSChop* **by** *blast*
have *4*: $\vdash$ *f*$\frown$ ($\neg$ (*sfin* (*init w*))) $\wedge$ *finite* $\longrightarrow$ ($\neg$ (*sfin* (*init w*)))
using *1 2 3* **by** *fastforce*
hence *5*: $\vdash$ *sfin* (*init w*) $\longrightarrow$ $\neg$ (*f*$\frown$ ($\neg$ (*sfin* (*init w*))))
by (*metis* (*no-types*, *lifting*) *DiamondFin SChopImpDiamond int-simps*(*32*) *int-simps*(*4*)
inteq-reflection sfin-d-def)
from *5* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SChopAndSFin*:
$\vdash$ ((*f*$\frown$ *g*) $\wedge$ (*sfin* (*init w*))) = *f*$\frown$ (*g* $\wedge$ (*sfin* (*init w*)))
proof −
have *1*: $\vdash$ *sfin* (*init w*) $\longrightarrow$ *f syields* (*sfin* (*init w*))
by (*rule SFinImpSYieldsSFin*)
have *2*: $\vdash$ (*f*$\frown$ *g*) $\wedge$ (*sfin* (*init w*)) $\longrightarrow$ (*f*$\frown$ *g*) $\wedge$ *f syields* (*sfin* (*init w*))
using *1* **by** *fastforce*
have *3*: $\vdash$ *f*$\frown$ *g* $\wedge$ *f syields* (*sfin* (*init w*)) $\longrightarrow$
f$\frown$ (*g* $\wedge$ (*sfin* (*init w*)))
using *SChopAndSYieldsImp* **by** *blast*
have *4*: $\vdash$ (*f*$\frown$ *g*) $\wedge$ (*sfin* (*init w*))$\longrightarrow$ *f*$\frown$ (*g* $\wedge$ *sfin* (*init w*))
using *2 3* **by** (*metis* (*mono-tags*, *lifting*) *lift-imp-trans*)
from *4* **show** *?thesis*
by (*simp add*: *Prop12 SChopAndA SChopSFinExportA int-iffI*)
qed

lemma *SChopAndNotSFin*:
$\vdash$ (*f*$\frown$ *g* $\wedge$ $\neg$ (*sfin* (*init w*)) $\wedge$ *finite*) = *f*$\frown$ (*g* $\wedge$ $\neg$ (*sfin* (*init w*)) $\wedge$ *finite*)
proof −
have *1*: $\vdash$ (*f*$\frown$ *g* $\wedge$ *sfin* (*init* ($\neg$ *w*))) = *f*$\frown$ (*g* $\wedge$ *sfin* (*init* ($\neg$ *w*)))
by (*rule SChopAndSFin*)
have *2*: $\vdash$ (*sfin* (*init* ($\neg$ *w*)) $\wedge$ *finite*)= (($\neg$ (*sfin* (*init w*))) $\wedge$ *finite*)
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
hence *3*: $\vdash$ (*g* $\wedge$ *sfin* (*init* ($\neg$ *w*))) = (*g* $\wedge$ $\neg$(*sfin* (*init w*)) $\wedge$ *finite*)
using *DiamondEmptyEqvFinite SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond*
by *fastforce*
hence *4*: $\vdash$ *f*$\frown$ (*g* $\wedge$ *sfin* (*init* ($\neg$ *w*))) = *f* $\frown$ (*g* $\wedge$ $\neg$ (*sfin* (*init w*)) $\wedge$ *finite*)
using *RightSChopEqvSChop* **by** *blast*
from *1 2 4* **show** *?thesis*
by (*metis DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqv-Diamond*

inteq-reflection)
qed

lemma *SFinSChopChain*:
$\vdash \quad (((init\ w\ )\ \wedge finite \longrightarrow\ sfin\ \ (init\ w1)))\frown$
$\quad (((init\ w1)\ \wedge finite \longrightarrow\ sfin\ \ (init\ w2)))$
$\quad \wedge finite$
$\longrightarrow (((init\ w\ )\ \wedge finite \longrightarrow\ sfin\ \ (init\ w2)))$
proof −
have *01*: $\vdash (init\ w \wedge finite \wedge$
$((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))) =$
$(init\ w \wedge empty) \frown (((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$
by (*meson Prop04 SChopAndCommute StateAndEmptySChop*)
have *02*: $\vdash (finite \wedge init\ w) = (init\ w \wedge empty) \frown finite$
by (*metis StateAndEmptySChop inteq-reflection lift-and-com*)
have *03*: $\vdash init\ w \wedge finite \wedge$
$((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow finite \wedge init\ w$
by *force*
have *04*: $\vdash (init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) =$
$(init\ w \wedge (finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)))$
by (*simp add*: *StateAndChop*)
have *05*: $\vdash init\ w \wedge finite \wedge ((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$
$(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$
by (*metis 04 AndChopCommute ChopAndB StateAndEmptyChop int-eq*)
have *06*: $\vdash init\ w \wedge finite \wedge ((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$
$(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$
by (*meson 03 05 Prop12*)
have *1*: $\vdash \ (init\ w) \wedge finite \wedge$
$((init\ w) \wedge finite \longrightarrow\ sfin\ \ (init\ w1))\frown$
$((init\ w1) \wedge finite \longrightarrow\ sfin\ \ (init\ w2))$
$\longrightarrow$
$((init\ w) \wedge finite \wedge\ ((init\ w) \wedge finite \longrightarrow sfin\ \ (init\ w1)))\frown$
$(((init\ w1) \wedge finite\longrightarrow\ sfin\ \ (init\ w2)) \wedge finite)$
unfolding *schop-d-def* **using** *06 ChopAndFiniteDist* **by** *fastforce*
have *2*: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite\longrightarrow\ sfin\ \ (init\ w1)) \longrightarrow$
$sfin\ \ (init\ w1)$
by *auto*
have *3*: $\vdash ((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow\ sfin\ \ (init\ w1)))\frown$
$(((init\ w1) \wedge finite \longrightarrow\ sfin\ \ (init\ w2)) \wedge finite)$
$\longrightarrow$
$(\ sfin\ \ (init\ w1))\frown (((init\ w1) \wedge finite \longrightarrow\ sfin\ \ (init\ w2)) \wedge finite)$
using *2 LeftSChopImpSChop* **by** *blast*

have *4*: ⊢ (*sfin* (*init w1*))⌢ (((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))) =
◇((*init w1*) ∧ ((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*)))
using *SFinSChopEqvDiamond* **by** *blast*
have *41*: ⊢ ((*init w1*) ∧ *finite* ∧ ((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))) ⟶ *sfin* (*init w2*)
by *auto*
have *42*: ⊢ ◇((*init w1*) ∧ *finite* ∧((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))) ⟶ ◇(*sfin*(*init w2*))
using *41 DiamondImpDiamond* **by** *blast*
have *5*: ⊢ ◇(*sfin*(*init w2*)) ⟶ *sfin* (*init w2*)
using *DiamondSFinImpSFin* **by** *blast*
have *51*: ⊢ ◇(*init w1* ∧ *finite* ∧ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*))) ⟶ *sfin* (*init w2*)
using *42 5 lift-imp-trans* **by** *blast*
have *52*: ⊢ *init w* ∧ *finite* ∧ (*init w* ∧ *finite* ⟶ *sfin* (*init w1*)) ⌢ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*))
⟶ *sfin* (*init w1*) ⌢ ((*init w1* ∧ *finite* ⟶ *sfin* (*init w2*)) ∧ *finite*)
by (*meson 1 3 lift-imp-trans*)
have *53*: ⊢ *init w* ∧ *finite* ∧ (*init w* ∧ *finite* ⟶ *sfin* (*init w1*)) ⌢ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*))
⟶ ◇(*init w1* ∧ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*)) ∧ *finite*)
by (*metis 52 SFinSChopEqvDiamond inteq-reflection*)
have *6*: ⊢ (*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *sfin* (*init w1*))⌢
((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))
⟶ *sfin* (*init w2*)
by (*metis 42 5 53 inteq-reflection lift-and-com lift-imp-trans*)
from *6* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopRule*:
assumes ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *sfin* (*init w1*)
⊢ (*init w1*)∧ *f1* ∧ *finite* ⟶ *sfin* (*init w2*)
shows ⊢ (*init w*) ∧ (*f*⌢ *f1*) ∧ *finite* ⟶ *sfin* (*init w2*)
proof −
have *1*: ⊢ (*init w*) ∧ (*f*⌢ *f1*) ∧ *finite* ⟶ ((*init w*) ∧ *f*)⌢ (*f1* ∧ *finite*)
by (*metis ChopEmpty SChopAssoc StateAndSChopImport inteq-reflection schop-d-def*)
have *2*: ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *sfin* (*init w1*)
using *assms* **by** *auto*
have *21*: ⊢ (*init w* ∧ *f* ∧ *finite*);(*f1* ∧ *finite*) = ((*init w* ∧ *f*);*f1* ∧ *finite*)
by (*metis ChopAndFiniteDist StateAndChop StateAndSChop inteq-reflection schop-d-def*)
have *22*: ⊢ (*init w* ∧ *f* ∧ *finite*) = ((*init w* ∧ *f* ∧ *finite*) ∧ *sfin* (*init w1*))
using *2 Prop10* **by** *blast*
hence *3*: ⊢ ((*init w*) ∧ *f*)⌢ (*f1* ∧ *finite*) ⟶ (*sfin* (*init w1*))⌢ (*f1* ∧ *finite*)
using *21 22*
by (*metis* (*no-types*, *opaque-lifting*) *2 ChopEmpty EmptySChop SChopAssoc StateAndSChop*
StateAndSChopImpSChopRule int-eq schop-d-def)
have *4*: ⊢ (*sfin* (*init w1*))⌢ (*f1* ∧ *finite*) = ◇((*init w1*) ∧ *f1* ∧ *finite*)
by (*rule SFinSChopEqvDiamond*)
have *5*: ⊢ (*init w1*) ∧ *f1* ∧ *finite* ⟶ *sfin* (*init w2*)
using *assms* **by** *auto*
hence *6*: ⊢ ◇((*init w1*) ∧ *f1* ∧ *finite*) ⟶ ◇ (*sfin* (*init w2*))
by (*rule DiamondImpDiamond*)
have *7*: ⊢ ◇(*sfin* (*init w2*)) ⟶ *sfin* (*init w2*)
using *DiamondSFinImpSFin* **by** *blast*
from *1 3 4 6 7* **show** *?thesis* **by** *fastforce*

qed

lemma *SChopRep*:
assumes $\vdash$ $(init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge \ sfin\ \ (init\ w1)$
$\vdash$ $(init\ w1) \wedge g \wedge finite \longrightarrow g1$
shows $\vdash$ $(init\ w) \wedge ((f \frown g) \wedge finite) \longrightarrow (f1 \frown g1)$
proof −
have *1*: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow (f1 \wedge \ sfin\ \ (init\ w1))$
using *assms* **by** *auto*
hence *2*: $\vdash (init\ w) \wedge (f \frown (g \wedge finite)) \longrightarrow (f1 \wedge \ sfin\ \ (init\ w1)) \frown (g \wedge finite)$
by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop12 SChopSFinExportA SFinEqvTrueSChopAndEmpty StateAndChopImpChopRule StateAndEmptySChop TrueSChopEqvDiamond inteq-reflection schop-d-def*)
have *3*: $\vdash (f1 \ \wedge \ sfin\ \ (init\ w1)) \frown (g \wedge finite) = f1 \frown ((init\ w1) \wedge (g \wedge finite))$
using *AndSFinSChopEqvStateAndSChop* **by** *blast*
have *31*: $\vdash$ $(init\ w) \wedge ((f \frown g) \wedge finite) \longrightarrow f1 \frown ((init\ w1) \wedge (g \wedge finite))$
using *2 3* **by** (*metis ChopEmpty SChopAssoc inteq-reflection schop-d-def*)
have *4*: $\vdash (init\ w1) \wedge (g \wedge finite) \longrightarrow g1$
using *assms* **by** *auto*
hence *5*: $\vdash f1 \frown ((init\ w1) \wedge (g \wedge finite)) \longrightarrow f1 \frown g1$
using *RightSChopImpSChop* **by** *blast*
show *?thesis* **using** *31 5* **by** *fastforce*
qed

lemma *SChopRepAndSFin*:
assumes $\vdash$ $(init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge \ sfin\ \ (init\ w1)$
$\vdash$ $(init\ w1) \wedge g \wedge finite \longrightarrow g1 \wedge \ sfin\ \ (init\ w2)$
shows $\vdash$ $(init\ w) \wedge (f \frown g) \wedge finite \longrightarrow (f1 \frown g1) \wedge \ sfin\ \ (init\ w2)$
proof −
have *1*: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge \ sfin\ \ (init\ w1)$
using *assms* **by** *auto*
have *2*: $\vdash (init\ w1) \wedge g \wedge finite \longrightarrow g1 \wedge \ sfin\ \ (init\ w2)$
using *assms* **by** *auto*
have *3*: $\vdash (init\ w) \wedge (f \frown g) \wedge finite \longrightarrow f1 \frown (g1 \wedge \ sfin\ \ (init\ w2))$
using *1 2* **by** (*rule SChopRep*)
have *4*: $\vdash f1 \frown (g1 \wedge \ sfin\ \ (init\ w2)) \longrightarrow f1 \frown g1$
by (*rule SChopAndA*)
have *5*: $\vdash f1 \frown (g1 \wedge \ sfin\ \ (init\ w2)) \longrightarrow f1 \frown \ sfin\ \ (init\ w2)$
by (*rule SChopAndB*)
have *6*: $\vdash f1 \frown \ sfin\ \ (init\ w2) \longrightarrow \ sfin\ \ (init\ w2)$
by (*rule SChopSFinImpSFin*)
show *?thesis*
by (*metis 3 4 5 6 Prop12 lift-imp-trans*)
qed

lemma *TrueSChopMoreEqvMore*:
$\vdash \#True \frown more = more$
by (*metis ChopAssoc TrueChopMoreEqvMore TrueEqvTrueSChopTrue inteq-reflection schop-d-def*)

lemma *SChopFmoreEqvFmore*:

$\vdash \#True \frown fmore = fmore$
by (*simp add*: *FiniteChopFmoreEqvFmore schop-d-def*)

lemma *MoreSChopLoop*:
assumes $\vdash \ f \longrightarrow \ more \frown f$
shows $\vdash finite \longrightarrow \neg \ f$
proof −
have *1*: $\vdash f \longrightarrow \ more \frown f$
using *assms* **by** *auto*
hence *11*: $\vdash \Diamond (f) \longrightarrow \Diamond (more \frown f)$
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash \Diamond (more \frown f) = \#True \frown (more \frown f)$
by (*simp add*: *DiamondSChopdef*)
have *13*: $\vdash \#True \frown (more \frown f) = (\#True \frown more) \frown f$
by (*rule SChopAssoc*)
have *14*: $\vdash \ \Diamond (more \frown f) = more \frown f$
using *12 13* **by** (*metis TrueSChopMoreEqvMore inteq-reflection*)
have *2*: $\vdash \ more \frown f = \bigcirc(\Diamond f)$
using *MoreSChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash \ \Diamond (f) \longrightarrow \bigcirc(\Diamond f)$
using *11 14 2* **by** *fastforce*
hence *4*: $\vdash finite \longrightarrow \neg (\Diamond f)$
using *NextLoop* **by** *blast*
have *5*: $\vdash \neg (\Diamond f) \longrightarrow \neg f$
using *NowImpDiamond* **by** *fastforce*
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *MoreSChopContra*:
assumes $\vdash \ f \wedge \neg \ g \longrightarrow (\ more \frown (f \wedge \neg \ g))$
shows $\vdash f \ \wedge finite \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg \ g \longrightarrow (\ more \frown (f \wedge \neg \ g))$ **using** *assms* **by** *auto*
hence *2*: $\vdash finite \longrightarrow \neg (f \wedge \neg \ g)$ **by** (*rule MoreSChopLoop*)
from *2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreSChopLoopFinite*:
assumes $\vdash \ f \wedge finite \longrightarrow \ more \frown f$
shows $\vdash finite \longrightarrow \neg \ f$
proof −
have *1*: $\vdash f \wedge finite \longrightarrow \ more \frown f$
using *assms* **by** *auto*
hence *11*: $\vdash \Diamond (f \wedge finite) \longrightarrow \Diamond (more \frown f)$
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash \Diamond (more \frown f) = \#True \frown (more \frown f)$
by (*simp add*: *DiamondSChopdef*)
have *13*: $\vdash \#True \frown (more \frown f) = (\#True \frown more) \frown f$
by (*rule SChopAssoc*)
have *14*: $\vdash \ \Diamond (more \frown f) = more \frown f$

using *12 13* **by** (*metis TrueSChopMoreEqvMore inteq-reflection*)
have *2*: $\vdash$ *more* $\frown$ *f* = $\bigcirc(\Diamond\ f)$
using *MoreSChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash$ $\Diamond\ (f \wedge \mathit{finite}) \longrightarrow \bigcirc(\Diamond\ f)$
using *11 14 2* **by** *fastforce*
have *31*: $\vdash \Diamond\ (f \wedge \mathit{finite}) = ((\Diamond\ f) \wedge \mathit{finite})$
by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SFinAndSChop SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection lift-and-com*)
have *32*: $\vdash (\Diamond\ f) \wedge \mathit{finite} \longrightarrow \bigcirc(\Diamond\ f)$
using *3 31* **by** *fastforce*
hence *4*: $\vdash \mathit{finite} \longrightarrow \neg\ (\Diamond\ f)$
by (*metis* (*no-types*, *lifting*) *DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09 finite-d-def int-simps*(*15*) *int-simps*(*32*) *inteq-reflection sometimes-d-def*)
have *5*: $\vdash \neg\ (\Diamond\ f) \longrightarrow \neg\ f$
by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreSChopContraFinite*:
assumes $\vdash$ $(f \wedge \neg\ g) \wedge \mathit{finite} \longrightarrow (\ \mathit{more} \frown (f \wedge \neg\ g))$
shows $\vdash f \wedge \mathit{finite} \longrightarrow g$
proof −
have *1*: $\vdash (f \wedge \neg\ g) \wedge \mathit{finite} \longrightarrow (\ \mathit{more} \frown (f \wedge \neg\ g))$ **using** *assms* **by** *auto*
hence *2*: $\vdash \mathit{finite} \longrightarrow \neg\ (f \wedge \neg\ g)$ **by** (*simp add*: *MoreSChopLoopFinite*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *SChopLoop*:
assumes $\vdash$ $f \longrightarrow g \frown f$
$\vdash g \longrightarrow \mathit{fmore}$
shows $\vdash \mathit{finite} \longrightarrow \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow g \frown f$ **using** *assms* **by** *auto*
have *2*: $\vdash g \longrightarrow \mathit{more}$ **using** *assms* **by** (*simp add*: *Prop12 fmore-d-def*)
hence *3*: $\vdash g \frown f \longrightarrow \mathit{more} \frown f$ **by** (*rule LeftSChopImpSChop*)
have *4*: $\vdash f \longrightarrow \mathit{more} \frown f$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreSChopLoop* **by** *auto*
qed

lemma *SChopLoopB*:
assumes $\vdash$ $f \longrightarrow g \frown f$
$\vdash g \longrightarrow \mathit{more}$
shows $\vdash \mathit{finite} \longrightarrow \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow g \frown f$ **using** *assms* **by** *auto*
have *2*: $\vdash g \longrightarrow \mathit{more}$ **using** *assms* **by** *auto*
hence *3*: $\vdash g \frown f \longrightarrow \mathit{more} \frown f$ **by** (*rule LeftSChopImpSChop*)
have *4*: $\vdash f \longrightarrow \mathit{more} \frown f$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreSChopLoop* **by** *blast*
qed

lemma *SChopContra*:
assumes $\vdash\ f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg\ (h \frown g)$
$\vdash\ h \longrightarrow\ fmore$
shows $\vdash f\ \wedge finite \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg\ (h \frown g)$ **using** *assms* **by** *auto*
have *2*: $\vdash h \longrightarrow\ more$ **using** *assms* **by** (*simp add*: *Prop12 fmore-d-def*)
have *3*: $\vdash h \frown f \wedge \neg\ (h \frown g) \longrightarrow h \frown (f \wedge \neg\ g)$ **by** (*rule SChopAndNotSChopImp*)
have *4*: $\vdash h \frown (f \wedge \neg\ g) \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *2* **by** (*rule LeftSChopImpSChop*)
have *5*: $\vdash f \wedge \neg\ g \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreSChopContra* **by** *auto*
qed

lemma *SChopContraB*:
assumes $\vdash f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg\ (h \frown g)$
$\vdash\ h \longrightarrow\ more$
shows $\vdash f \wedge finite \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg\ (h \frown g)$ **using** *assms* **by** *auto*
have *2*: $\vdash h \longrightarrow\ more$ **using** *assms* **by** *auto*
have *3*: $\vdash h \frown f \wedge \neg\ (h \frown g) \longrightarrow h \frown (f \wedge \neg\ g)$ **by** (*rule SChopAndNotSChopImp*)
have *4*: $\vdash h \frown (f \wedge \neg\ g) \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *2* **by** (*rule LeftSChopImpSChop*)
have *5*: $\vdash f \wedge \neg\ g \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreSChopContra* **by** *blast*
qed

3.4.7 Properties of Halt

lemma *HaltSChopEqv*:
$\vdash\ ((halt\ (\ init\ w)) \frown f) =\ (if_i\ \ (init\ w)\ \ then\ (\ f\ )\ else\ (\bigcirc(\ (halt\ (\ init\ w)) \frown f)))$
proof −
have *1*: $\vdash\ halt(init\ w) =$
$(if_i\ \ (init\ w)\ \ then\ \ \ empty\ \ \ else\ (\ \bigcirc(\ halt\ (\ init\ w))))$
by (*rule HaltStateEqvIfStateThenEmptyElseNext*)
hence *2*: $\vdash\ ((halt(init\ w)) \frown f) =$
$(if_i\ \ (init\ w)\ \ then\ \ \ (empty \frown f)\ \ \ \ else\ (\ \bigcirc(\ halt\ (\ init\ w)) \frown f))$
by (*rule IfSChopEqvRule*)
have *3*: $\vdash\ empty \frown f = f$
by (*rule EmptySChop*)
have *4*: $\vdash (\bigcirc\ (halt\ (\ init\ w))) \frown f = \bigcirc(\ halt\ (\ init\ w) \frown f)$
by (*rule NextSChop*)
from *2 3 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *AndHaltSChopImp*:
$\vdash\ \ init\ w \wedge (\ halt\ (\ init\ w) \frown f) \longrightarrow f$
proof −
have *1*: $\vdash halt\ (\ init\ w) \frown f =\ \ if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w) \frown f))$
by (*rule HaltSChopEqv*)

have *2*: $\vdash$ *init w* $\wedge$ *if*$_i$ (*init w*) *then f else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$ *f*
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltSChopImpNext*:
$\vdash$ $\neg$ (*init w*) $\wedge$ (*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ $\bigcirc$(*halt* (*init w*)$\frown$ *f*)
proof −
have *1*: $\vdash$ *halt* (*init w*)$\frown$ *f* = *if*$_i$ (*init w*) *then f else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))
by (*rule HaltSChopEqv*)
have *2*: $\vdash$ $\neg$ (*init w*) $\wedge$ *if*$_i$ (*init w*) *then f else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$(*halt* (*init w*)$\frown$ *f*)
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltSChopImpSkipSYields*:
$\vdash$ $\neg$ (*init w*) $\wedge$ (*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ *skip syields* (*halt* (*init w*)$\frown$ *f*)
proof −
have *1*: $\vdash$ $\neg$ (*init w*) $\wedge$ (*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ $\bigcirc$(*halt* (*init w*)$\frown$ *f*)
by (*rule NotAndHaltSChopImpNext*)
have *2*: $\vdash$ $\bigcirc$(*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ *skip syields* (*halt* (*init w*)$\frown$ *f*)
by (*rule NextImpSkipSYields*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndEmptyEqvSChopAndEmpty*:
$\vdash$ ((#*True*$\frown$(*f* $\wedge$ *empty*)) $\wedge$ *g*) = (*g*$\frown$(*f*$\wedge$ *empty*))
proof −
have *1*: $\vdash$ (#*True*$\frown$(*f* $\wedge$ *empty*)) $\wedge$ *g* $\longrightarrow$ *g* $\frown$(*f* $\wedge$ *empty*)
by (*simp add*: *FiniteChopAndEmptyEqvChopAndEmpty int-iffD1 schop-d-def*)
have *2*: $\vdash$ *g*$\frown$(*f*$\wedge$ *empty*) $\longrightarrow$ (#*True*$\frown$(*f* $\wedge$ *empty*)) $\wedge$ *g*
by (*metis AndSFinEqvSChopAndEmpty Prop12 SFinEqvTrueSChopAndEmpty int-iffD1 inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotSChopSkipEqvFmoreAndNotSChopSkip*:
$\vdash$ ($\neg$ *f*)$\frown$*skip* = (*fmore* $\wedge$ $\neg$(*f*$\frown$*skip*))
proof −
have *1*: $\vdash$ ($\neg$ *f*)$\frown$*skip* = (($\neg$*f* $\wedge$ *finite*);*skip*)
by (*simp add*: *schop-d-def*)
have *2*: $\vdash$ ($\neg$*f* $\wedge$ *finite*);*skip* = ($\neg$(*f* $\vee$ *inf*));*skip*
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop finite-d-def int-simps*(*14*) *int-simps*(*33*)
inteq-reflection)
have *3*: $\vdash$ ($\neg$(*f* $\vee$ *inf*));*skip* = (*more* $\wedge$ $\neg$((*f* $\vee$ *inf*);*skip*))
using *NotChopSkipEqvMoreAndNotChopSkip* **by** *blast*
have *4*: $\vdash$ (*f* $\vee$ *inf*);*skip* = (*f*;*skip* $\vee$ *inf*)
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv inteq-reflection*)
have *5*: $\vdash$ (*more* $\wedge$ $\neg$((*f* $\vee$ *inf*);*skip*)) = (*more* $\wedge$ $\neg$(*f*;*skip* $\vee$ *inf*))
using *4* **by** *auto*

have *6*: $\vdash$ $(more \wedge \neg(f;skip \vee inf)) = (more \wedge \neg(f;skip) \wedge finite)$
unfolding *finite-d-def* **by** *fastforce*
have *7*: $\vdash$ $(more \wedge \neg(f;skip) \wedge finite) = (more \wedge \neg(f \frown skip \vee (f \wedge inf)) \wedge finite)$
by (*metis ChopEmpty ChopSChopdef inteq-reflection*)
have *8*: $\vdash$ $(more \wedge \neg(f \frown skip \vee (f \wedge inf)) \wedge finite) =$
$(more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite)$
by *auto*
have *9*: $\vdash$ $(\neg(f \wedge inf) \wedge finite) = finite$
unfolding *finite-d-def* **by** *force*
have *10*: $\vdash$ $(more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite) =$
$(more \wedge \neg(f \frown skip) \wedge finite)$
using *9* **by** *fastforce*
have *11*: $\vdash$ $(more \wedge \neg(f \frown skip) \wedge finite) = (fmore \wedge \neg(f \frown skip))$
using *fmore-d-def* **by** (*metis Prop11 Prop12 lift-and-com*)
from *1 2 3 5 6 7 8 10 11* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *HaltSChopImpNotHaltSChopNot*:
$\vdash$ $halt\ (\ init\ w) \frown f \wedge finite \longrightarrow \neg\ (\ halt\ (\ init\ w) \frown (\neg\ f))$
proof −
have *1*: $\vdash$ $halt\ \ (init\ w) \frown f =\ \ if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w) \frown f))$
by (*rule HaltSChopEqv*)
have *2*: $\vdash$ $if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w) \frown f)) \longrightarrow$
$(\ ((init\ w) \longrightarrow f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown f))))$
by (*rule IfThenElseImp*)
have *3*: $\vdash$ $halt\ \ (init\ w) \frown (\neg f) =$
$if_i\ \ (init\ w)\ \ then\ \ (\neg f)\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))$
by (*rule HaltSChopEqv*)
have *4*: $\vdash$ $if_i\ \ (init\ w)\ \ then\ \ (\neg f)\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f))) \longrightarrow$
$(\ ((init\ w) \longrightarrow \neg f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))))$
by (*rule IfThenElseImp*)
have *5*: $\vdash$ $halt\ \ (init\ w) \frown f \wedge halt\ \ (init\ w) \frown (\neg f) \longrightarrow$
$(\ ((init\ w) \longrightarrow f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown f)))) \wedge$
$(\ ((init\ w) \longrightarrow \neg f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))))$
using *1 2 3 4* **by** *fastforce*
have *6*: $\vdash$ $(\ ((init\ w) \longrightarrow f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown f)))) \wedge$
$(\ ((init\ w) \longrightarrow \neg f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f))))) \longrightarrow$
$(\ \bigcirc(\ halt\ (\ init\ w) \frown f)) \wedge (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))$
by *auto*
have *7*: $\vdash$ $halt\ \ (init\ w) \frown f \wedge halt\ \ (init\ w) \frown (\neg f) \longrightarrow$
$(\ \bigcirc(\ halt\ (\ init\ w) \frown f)) \wedge (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))$
using *5 6 lift-imp-trans* **by** *blast*
have *8*: $\vdash$ $((\ \bigcirc(\ halt\ (\ init\ w) \frown f)) \wedge (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))) =$
$\bigcirc\ (halt\ (\ init\ w) \frown f \wedge halt\ (\ init\ w) \frown (\neg f))$
using *NextAndEqvNextAndNext* **by** *fastforce*
have *9*: $\vdash$ $halt\ \ (init\ w) \frown f \wedge halt\ \ (init\ w) \frown (\neg f) \longrightarrow$
$\bigcirc\ (halt\ (\ init\ w) \frown f \wedge halt\ (\ init\ w) \frown (\neg f))$
using *7 8* **by** *fastforce*
hence *10*: $\vdash$ $finite \longrightarrow \neg(halt\ \ (init\ w) \frown f \wedge halt\ \ (init\ w) \frown (\neg f))$
using *NextLoop* **by** *blast*

from *10* **show** *?thesis* **by** *auto*
qed

lemma *HaltSChopImpHaltSYields*:
$\vdash$ *halt* (*init w*)$^\frown$ *f* $\wedge$ *finite* $\longrightarrow$ (*halt* (*init w*)) *syields* *f*
proof −
have *1*: $\vdash$ *halt* (*init w*)$^\frown$ *f* $\wedge$ *finite* $\longrightarrow$ $\neg$ (*halt* (*init w*)$^\frown$ ($\neg$ *f*))
by (*rule HaltSChopImpNotHaltSChopNot*)
from *1* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *HaltSChopAnd*:
$\vdash$ (*halt* (*init w*))$^\frown$ *f* $\wedge$ (*halt* (*init w*))$^\frown$ *g* $\wedge$ *finite*$\longrightarrow$ (*halt* (*init w*))$^\frown$ (*f* $\wedge$ *g*)
proof −
have *1*: $\vdash$ (*halt* (*init w*))$^\frown$ *g* $\wedge$ *finite* $\longrightarrow$ (*halt* (*init w*)) *syields* *g*
by (*rule HaltSChopImpHaltSYields*)
hence *2*: $\vdash$ (*halt* (*init w*))$^\frown$ *f* $\wedge$ (*halt* (*init w*))$^\frown$ *g* $\wedge$ *finite* $\longrightarrow$
(*halt* (*init w*))$^\frown$ *f* $\wedge$ (*halt* (*init w*)) *syields* *g*
by *auto*
have *3*: $\vdash$ (*halt* (*init w*))$^\frown$ *f* $\wedge$ (*halt* (*init w*)) *syields* *g*$\longrightarrow$
(*halt* (*init w*))$^\frown$ (*f* $\wedge$ *g*)
by (*rule SChopAndSYieldsImp*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltAndSChopAndHaltSChopImpHaltAndSChopAnd*:
$\vdash$ (*halt* (*init w*) $\wedge$ *f*)$^\frown$ *f1* $\wedge$ (*halt* (*init w*)$^\frown$ *g*) $\wedge$ *finite*
$\longrightarrow$ (*halt* (*init w*) $\wedge$ *f*)$^\frown$ (*f1* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *f1* $\longrightarrow$ $\neg$ *g* $\vee$ (*f1* $\wedge$ *g*)
by *auto*
hence *2*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$^\frown$ *f1* $\longrightarrow$
(*halt* (*init w*) $\wedge$ *f*)$^\frown$ ($\neg$ *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*)$^\frown$ (*f1* $\wedge$ *g*))
by (*rule SChopOrImpRule*)
have *3*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$^\frown$ ($\neg$ *g*) $\longrightarrow$ *halt* (*init w*)$^\frown$ ($\neg$ *g*)
by (*rule AndSChopA*)
have *31*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$^\frown$ *f1* $\longrightarrow$
halt (*init w*)$^\frown$ ($\neg$ *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*)$^\frown$ (*f1* $\wedge$ *g*))
using *23* **by** *fastforce*
have *4*: $\vdash$ *halt* (*init w*)$^\frown$ *g* $\wedge$ *finite* $\longrightarrow$ $\neg$ (*halt* (*init w*)$^\frown$ ($\neg$ *g*))
by (*rule HaltSChopImpNotHaltSChopNot*)
hence *41*: $\vdash$ (*halt* (*init w*)$^\frown$ ($\neg$ *g*)) $\wedge$ *finite* $\longrightarrow$ $\neg$(*halt* (*init w*)$^\frown$ *g*)
by *auto*
have *42*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$^\frown$ *f1* $\wedge$ *finite* $\longrightarrow$
$\neg$(*halt* (*init w*)$^\frown$ *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*)$^\frown$ (*f1* $\wedge$ *g*))
using *31 41* **by** *fastforce*
from *42* **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxSYields*:

$\vdash$ $(\ halt\ \ (init\ w))\frown f \wedge finite \longrightarrow (\Box(\neg\ (\ init\ w)))$ *syields* $((\ halt\ \ (init\ w))\frown f)$

proof −

have *1*: $\vdash (\Box\ (\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)) \longrightarrow\ df\ (\Box\ (\neg\ \ (init\ w)))$
by (*rule SChopImpDf*)

have *2*: $\vdash \Box\ (\neg\ \ (init\ w)) \longrightarrow \neg\ \ (init\ w)$
by (*rule BoxElim*)

hence *3*: $\vdash\ df\ (\Box\ (\neg\ \ (init\ w))) \longrightarrow\ df\ (\neg\ \ (init\ w))$
by (*rule DfImpDf*)

have *4*: $\vdash\ df\ \ (init\ (\neg\ w)) =\ \ (init\ (\neg w))$
by (*rule DfState*)

have *41*: $\vdash\ (init\ (\neg\ w)) =\ (\neg\ (init\ w))$
using *Initprop*(*2*) **by** *fastforce*

have *42*: $\vdash\ df\ \ (\neg\ (init\ \ w)) =\ \ (\neg(init\ w))$
using *4 41* **by** (*metis inteq-reflection*)

have *5*: $\vdash ((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f))) \longrightarrow \neg\ (\ init\ w)$
using *1 2 42* **using** *3* **by** *fastforce*

hence *51*: $\vdash (\ halt\ \ (init\ w)\frown f) \wedge ((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f))) \longrightarrow$
$(\ halt\ \ (init\ w)\frown f) \wedge \neg\ (\ init\ w)$
by *fastforce*

have *6*: $\vdash\ halt\ \ (init\ w)\frown f = if_i\ (init\ w)\ \ then\ \ f\ \ else\ \ (\bigcirc(\ halt\ \ (init\ w)\frown f))$
by (*rule HaltSChopEqv*)

hence *61*: $\vdash (halt\ \ (init\ w)\frown f \wedge \neg\ (\ init\ w)) =$
$((if_i\ (init\ w)\ \ then\ \ f\ \ else\ \ (\bigcirc(\ halt\ \ (init\ w)\frown f))) \wedge \neg\ (\ init\ w))$
using *6* **by** *auto*

have *62*: $\vdash (if_i\ (init\ w)\ \ then\ \ f\ \ else\ \ (\bigcirc(\ halt\ \ (init\ w)\frown f))) \wedge$
$\neg\ (\ init\ w) \longrightarrow (\bigcirc(\ halt\ \ (init\ w)\frown f))$
by (*auto simp*: *ifthenelse-d-def*)

have *63*: $\vdash halt\ \ (init\ w)\frown f \wedge \neg\ (\ init\ w) \longrightarrow (\bigcirc(\ halt\ \ (init\ w)\frown f))$
using *61 62* **by** *fastforce*

have *7*: $\vdash (\ halt\ \ (init\ w)\frown f) \wedge (\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)) \longrightarrow$
$\bigcirc((\ halt\ \ (init\ w))\frown f)$
using *51 63* **using** *lift-imp-trans* **by** *blast*

have *8*: $\vdash \Box\ (\neg\ (init\ w)) \longrightarrow\ empty\ \vee\ \bigcirc(\Box(\neg(\ \ init\ w)))$
by (*metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq*)

hence *9*: $\vdash ((\Box\ (\neg\ (\ init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f))) \longrightarrow$
$\neg\ (\ halt\ \ (init\ w)\frown f) \vee\ \bigcirc((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)))$
by (*rule EmptyOrNextSChopImpRule*)

hence *10*: $\vdash ((\ halt\ \ (init\ w))\frown f) \wedge (\Box\ (\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)) \longrightarrow$
$\bigcirc((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)))$
by *fastforce*

have *11*: $\vdash (\ halt\ \ (init\ w))\frown f \wedge (\Box\ (\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)) \longrightarrow$
$\bigcirc((\ halt\ \ (init\ w))\frown f) \wedge \bigcirc((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)))$
using *7 10* **by** *fastforce*

have *12*: $\vdash \bigcirc((\ halt\ \ (init\ w))\frown f) \wedge \bigcirc((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)))$
$\longrightarrow\ \bigcirc(((\ halt\ (init\ w))\frown f) \wedge ((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f))))$
using *NextAndEqvNextAndNext* **by** *fastforce*

have *13*: $\vdash (\ halt\ \ (init\ w))\frown f \wedge (\Box\ (\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f)) \longrightarrow$
$\bigcirc(((\ halt\ \ (init\ w))\frown f) \wedge ((\Box(\neg\ \ (init\ w)))\frown (\neg\ (\ halt\ \ (init\ w)\frown f))))$
using *11 12* **by** *fastforce*

hence *14*: $\vdash finite \longrightarrow \neg\ ((\ halt\ \ (init\ w))\frown f \wedge (\Box\ (\neg\ (init\ w)))\frown (\neg(halt\ (init\ w)\frown f)))$

using *NextLoop* **by** *blast*
hence *15*: $\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ *finite* $\longrightarrow$ $\neg$ (($\Box$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
by *auto*
from *15* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

3.4.8 Properties of Groups of strong chops

lemma *NestedSChopImpSChop*:
assumes $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g* $\frown$ (*init w1* $\wedge$ *f1*)
$\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*$\frown$ (*init w2* $\wedge$ *f2*)
shows $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g* $\frown$ (*g1*$\frown$ (*init w2* $\wedge$ *f2*))
proof −
have *1*: $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*$\frown$ (*init w1* $\wedge$ *f1*) **using** *assms*(*1*) **by** *auto*
have *2*: $\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*$\frown$ (*init w2* $\wedge$ *f2*) **using** *assms*(*2*) **by** *auto*
hence *3*: $\vdash$ *g*$\frown$ (*init w1* $\wedge$ *f1*) $\longrightarrow$ *g*$\frown$ (*g1*$\frown$ (*init w2* $\wedge$ *f2*)) **by** (*rule RightSChopImpSChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

end

3.5 Chopstar and variants

theory *Chopstar*
imports
SChopTheorems
begin

This theory defines the chopstar operator for infinite ITL and provides a library of lemmas. We also define the strong version schopstar, the weak version wpowerstar and a semantic version achopstar. The wpowerstar corresponds to the Kleene star operator from Kleene Algebra [1]. We provide lemmas that express various relationships between them. We also ported the numerous Kleene algebra lemmas from [1] to ITL.

3.5.1 Definitions

primrec *wpower-d* :: ('*a*::*world*) *formula* $\Rightarrow$ *nat* $\Rightarrow$ '*a formula*
where *wpow-0* : (*wpower-d F 0*) = *LIFT*(*empty*)
| *wpow-Suc*: (*wpower-d F* (*Suc n*)) = *LIFT*((*F*);(*wpower-d F n*))

syntax
-wpower-d :: [*lift*,*nat*] $\Rightarrow$ *lift* ((*wpower - -*) [*88*,*88*] *87*)

syntax (*ASCII*)
-wpower-d :: [*lift*,*nat*] $\Rightarrow$ *lift* ((*wpower - -*) [*88*,*88*] *87*)

translations
-wpower-d $\rightleftharpoons$ *CONST wpower-d*

definition *fpower-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *nat* $\Rightarrow$ *'a formula*
where
fpower-d F n $\equiv$ *LIFT*(*wpower* (*F* $\wedge$ *finite*) *n*)

syntax
-fpower-d :: [*lift*,*nat*] $\Rightarrow$ *lift* ((*fpower* - -) [*88*,*88*] *87*)

syntax (*ASCII*)
-fpower-d :: [*lift*,*nat*] $\Rightarrow$ *lift* ((*fpower* - -) [*88*,*88*] *87*)

translations
-fpower-d $\rightleftharpoons$ *CONST fpower-d*

definition *len-d* :: *nat* $\Rightarrow$ (*'a*::*world*) *formula*
where *len-d n* $\equiv$ *LIFT*(*wpower skip n*)

definition *wpowerstar-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula*
where *wpowerstar-d F* $\equiv$ *LIFT*(($\exists$ *k*. *wpower F k*))

definition *fpowerstar-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula*
where *fpowerstar-d F* $\equiv$ *LIFT*($\exists$ *k*. *fpower F k*)

syntax
-len-d :: *nat* $\Rightarrow$ *lift* ((*len* -) [*88*] *87*)
-wpowerstar-d :: *lift* $\Rightarrow$ *lift* ((*wpowerstar* -) [*85*] *85*)
-fpowerstar-d :: *lift* $\Rightarrow$ *lift* ((*fpowerstar* -) [*85*] *85*)

syntax (*ASCII*)
-len-d :: *nat* $\Rightarrow$ *lift* ((*len* -) [*88*] *87*)
-wpowerstar-d :: *lift* $\Rightarrow$ *lift* ((*wpowerstar* -) [*85*] *85*)
-fpowerstar-d :: *lift* $\Rightarrow$ *lift* ((*fpowerstar* -) [*85*] *85*)

translations
-len-d $\rightleftharpoons$ *CONST len-d*
-wpowerstar-d $\rightleftharpoons$ *CONST wpowerstar-d*
-fpowerstar-d $\rightleftharpoons$ *CONST fpowerstar-d*

definition *powerstar-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula*
where *powerstar-d F* $\equiv$ *LIFT*(*fpowerstar F*;(*empty* $\vee$ (*F* $\wedge$ *inf*)))

syntax
-powerstar-d :: *lift* ⇒ *lift* ((*powerstar -*) [*85*] *85*)

syntax (*ASCII*)
-powerstar-d :: *lift* ⇒ *lift* ((*powerstar -*) [*85*] *85*)

translations
-powerstar-d ⇌ *CONST powerstar-d*

definition *chopstar-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *chopstar-d F* ≡ *LIFT*(*powerstar* (*F* ∧ *more*))

definition *schopstar-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *schopstar-d F* ≡ *LIFT*(*fpowerstar* (*F* ∧ *more*))

syntax
-chopstar-d :: *lift* ⇒ *lift* ((*-*⋆) [*85*] *85*)
-schopstar-d ::*lift* ⇒ *lift* ((*schopstar -*) [*85*] *85*)

syntax (*ASCII*)
-chopstar-d :: *lift* ⇒ *lift* ((*chopstar -*) [*85*] *85*)
-schopstar-d ::*lift* ⇒ *lift* ((*schopstar -*) [*85*] *85*)

translations
-chopstar-d ⇌ *CONST chopstar-d*
-schopstar-d ⇌ *CONST schopstar-d*

definition *while-d* :: ('*a*::*world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *while-d F G* ≡ *LIFT*((*F* ∧ *G*)⋆ ∧ (*fin* ((¬*F*))))

syntax
-while-d :: [*lift*,*lift*] ⇒ *lift* ((*while - do -*) [*88*,*88*] *87*)

syntax (*ASCII*)
-while-d :: [*lift*,*lift*] ⇒ *lift* ((*while - do -*) [*88*,*88*] *87*)

translations
-while-d ⇌ *CONST while-d*

definition *swhile-d* :: ('*a*::*world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *swhile-d F G* ≡ *LIFT*(*schopstar*(*F* ∧ *G*) ∧ (*sfin* ((¬*F*))))

definition *repeat-d* :: ('*a*::*world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *repeat-d F G* ≡ *LIFT*(*F*;*while* (¬ *G*) *do F*)

syntax
-swhile-d :: [*lift*,*lift*] ⇒ *lift* ((*swhile - do -*) [*88*,*88*] *87*)
-repeat-d :: [*lift*,*lift*] ⇒ *lift* ((*repeat - until -*) [*88*,*88*] *87*)

syntax (*ASCII*)
-swhile-d :: [*lift*,*lift*] ⇒ *lift* ((*swhile - do -*) [*88*,*88*] *87*)
-repeat-d :: [*lift*,*lift*] ⇒ *lift* ((*repeat - until -*) [*88*,*88*] *87*)

translations
-swhile-d ⇌ *CONST swhile-d*
-repeat-d ⇌ *CONST repeat-d*

definition *srepeat-d* :: ('*a*::*world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *srepeat-d F G* ≡ *LIFT*(*F*⌢*swhile* (¬ *G*) *do F*)

syntax
-srepeat-d :: [*lift*,*lift*] ⇒ *lift* ((*srepeat - until -*) [*88*,*88*] *87*)

syntax (*ASCII*)
-srepeat-d :: [*lift*,*lift*] ⇒ *lift* ((*srepeat - until -*) [*88*,*88*] *87*)

translations
-srepeat-d ⇌ *CONST srepeat-d*

definition *aschopstar-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *aschopstar-d F* ≡
λ*s*. (∃ (*l*:: *nat nellist*).
(*nnth l 0*) = *0* ∧ *nfinite l* ∧ *nidx l* ∧
(*enat* (*nlast l*)) = (*nlength s*) ∧ *nfinite s* ∧
(∀ (*i*::*nat*) . ((*enat i*)< (*nlength l*)) ⟶
((*nsubn s* (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *F*))
)

syntax
-aschopstar-d :: *lift* ⇒ *lift* ((*aschopstar -*) [*85*] *85*)

syntax (*ASCII*)
-aschopstar-d :: *lift* ⇒ *lift* ((*aschopstar -*) [*85*] *85*)

translations
-aschopstar-d ⇌ *CONST aschopstar-d*

lemma *FChopSem-var* [*mono*]:
(*w* ⊨ *f*;*g*) =
(((∃ *n*. *enat n* ≤ *nlength w* ∧ *f* ((*ntaken n w*)) ∧ *g* (*ndropn n w*)) ∨
(¬ *nfinite w* ∧ *f w*)))
by (*simp add*: *itl-defs*)

inductive *istar-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
for *F* **where**

$(s \models \mathit{empty}) \Longrightarrow (s \models (\mathit{istar\text{-}d}\ F))$
| $(s \models F;(\mathit{istar\text{-}d}\ F)) \Longrightarrow (s \models (\mathit{istar\text{-}d}\ F))$

syntax
$\mathit{-istar\text{-}d} :: \mathit{lift} \Rightarrow \mathit{lift}$ $((\mathit{istar}\ \text{-})\ [85]\ 85)$

syntax (*ASCII*)
$\mathit{-istar\text{-}d} :: \mathit{lift} \Rightarrow \mathit{lift}$ $((\mathit{istar}\ \text{-})\ [85]\ 85)$

translations
$\mathit{-istar\text{-}d}$ $\rightleftharpoons$ $\mathit{CONST\ istar\text{-}d}$

3.5.2 Semantic lemmas

lemma *ChopExist*:
$\vdash (\exists\ k.\ f;g\ k) = f;(\exists\ k.\ g\ k)$
by (*auto simp add*: *itl-defs Valid-def*)

lemma *SChopExist*:
$\vdash (\exists\ k.\ f \frown g\ k) = f \frown (\exists\ k.\ g\ k)$
by (*auto simp add*: *itl-defs Valid-def*)

lemma *ExistChop*:
$\vdash (\exists\ k.\ (g\ k);f) = (\exists\ k.\ g\ k);f$
by (*auto simp add*: *itl-defs Valid-def*)

lemma *ExistSChop*:
$\vdash (\exists\ k.\ (g\ k) \frown f) = (\exists\ k.\ g\ k) \frown f$
by (*auto simp add*: *itl-defs Valid-def*)

lemma *wpowersem1*:
$(\sigma \models (\exists\ k.\ \mathit{wpower}\ f\ k) = (\ \mathit{empty} \vee (\exists\ k.\ \mathit{wpower}\ f\ (\mathit{Suc}\ k))))$
proof (*auto*)
show $\bigwedge x.\ \sigma \models (\mathit{wpower}\ f\ x) \Longrightarrow \forall k.\ \neg\ (\sigma \models f;\mathit{wpower}\ f\ k) \Longrightarrow \sigma \models \mathit{empty}$
by (*metis not0-implies-Suc wpow-0 wpow-Suc*)
show $\sigma \models \mathit{empty} \Longrightarrow \exists x.\ \sigma \models (\mathit{wpower}\ f\ x)$
by (*metis wpow-0*)
show $\bigwedge k.\ \sigma \models (f;\mathit{wpower}\ f\ k) \Longrightarrow \exists x.\ \sigma \models (\mathit{wpower}\ f\ x)$
by (*metis wpow-Suc*)
qed

lemma *fpowersem1*:
$(\sigma \models (\exists\ k.\ \mathit{fpower}\ f\ k) = (\ \mathit{empty} \vee (\exists\ k.\ \mathit{fpower}\ f\ (\mathit{Suc}\ k))))$
unfolding *fpower-d-def* **using** *wpowersem1*[*of LIFT* $(f \wedge \mathit{finite})\ \sigma$]
by *blast*

lemma *wpowersem*:
$\vdash (\exists\ k.\ wpower\ f\ k) = (\ \ empty \vee f;(\exists\ k.\ (wpower\ f\ k)))$
proof −
have *1*: $\vdash (\exists\ k.\ wpower\ f\ k) = (\ empty \vee (\exists\ k.\ wpower\ f\ (Suc\ k)))$
using *wpowersem1* **by** *blast*
have *2*: $\vdash (\exists\ k.\ wpower\ f\ (Suc\ k)) = (\exists\ k.\ f;wpower\ f\ k)$
by *simp*
have *3*: $\vdash (\exists\ k.\ f;(wpower\ f\ k)) = f;(\exists\ k.\ (wpower\ f\ k))$
using *ChopExist* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *fpowersem*:
$\vdash (\exists\ k.\ fpower\ f\ k) = (\ \ empty \vee (f \wedge finite);(\exists\ k.\ (fpower\ f\ k)))$
unfolding *fpower-d-def* **using** *wpowersem*[*of LIFT* $(f \wedge finite)$]
by *blast*

lemma *finite-nidx-bounded-nlast*:
assumes *nfinite l*
$nidx\ l$
$(enat\ (nlast\ l)) = (nlength\ s)$
$(enat\ i) \leq (nlength\ l)$
shows $(nnth\ l\ i) \leq nlength\ s$
using *assms*
by (*metis enat-ord-simps*(*1*) *nidx-all-le-nlast*)

lemma *aschopstar-wpower-chain-a*:
assumes $(\exists\ (l::\ nat\ nellist).$
$(nlength\ l) = (Suc\ n) \wedge nfinite\ l \wedge nidx\ l \wedge (nnth\ l\ 0) = 0 \wedge$
$(enat\ (nlast\ l)) = (nlength\ \sigma) \wedge nfinite\ \sigma \wedge$
$(\forall (i::nat).\ i< (nlength\ l) \longrightarrow$
$((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models f)$
$)$
$)$
shows $(\exists\ k.\ 0 \leq k \wedge k \leq nlength\ \sigma \wedge 0< k \wedge (nsubn\ \sigma\ 0\ k\ \models f) \wedge$
$(\exists\ ls.\ nfinite\ ls \wedge (nlength\ ls) = n \wedge nidx\ ls \wedge (nnth\ ls\ 0) = 0 \wedge$
$(enat\ (nlast\ ls)) = (nlength\ (ndropn\ k\ \sigma)) \wedge nfinite(ndropn\ k\ \sigma) \wedge$
$(\forall (i::nat).\ \ i< (nlength\ ls) \longrightarrow$
$((nsubn\ (ndropn\ k\ \sigma)\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ ) \models f)$
$))$
$)$
proof −
obtain *l* **where** *1*: $(nlength\ l) = (Suc\ n) \wedge nfinite\ l \wedge\ \ nidx\ l \wedge (nnth\ l\ 0) = 0 \wedge$
$(enat\ (nlast\ l))\ \ = (nlength\ \sigma) \wedge nfinite\ \sigma \wedge$
$(\forall\ \ (i::nat).\ \ i< (nlength\ l) \longrightarrow$
$((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ ) \models f)$
$)$

using *assms* **by** *auto*
have *2*: *nlength l >0*
using *1* **by** (*simp add*: *enat-0-iff*(*1*))
have *3*: *l = NCons* (*nnth l 0*) (*ndropn 1 l*)
using *2* **by** (*simp add*: *ndropn-Suc-conv-ndropn zero-enat-def*)
have *4*: *nlength* (*ndropn 1 l*) = *n*
by (*simp add*: *1*)
have *5*: *0*≤(*nnth l 0*)
by *simp*
have *6*: (*nnth l 0*) ≤ *nlength σ*
using *1* **using** *i0-lb zero-enat-def* **by** *presburger*
have *7*: (*nnth l 0*) = *0*
using *1* **by** *blast*
have *71*: (*nnth* (*ndropn 1 l*) *0*) = (*nnth l 1*)
by *auto*
have *8*: *nnth l 0* < *nnth* (*ndropn 1 l*) *0*
by (*metis 1 71 One-nat-def eSuc-enat enat-ord-simps*(*2*) *ileI1 nidx-gr-first zero-less-Suc*)
have *9*: *nidx* (*ndropn 1 l*)
by (*metis 1 2 One-nat-def Suc-eq-plus1 Suc-ile-eq enat-min-eq ndropn-nlength ndropn-nnth nidx-expand plus-1-eq-Suc plus-enat-simps*(*1*) *zero-enat-def*)
have *10*: (*enat* (*nlast* (*ndropn 1 l*))) = (*nlength σ*)
using *1 3* **by** (*metis nlast-NCons*)
have *101*: ⋀*j*. *j* ≤ *nlength* (*ndropn 1 l*) ⟶
nnth (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*)) *j* =
(*nnth l* (*Suc j*)) − (*nnth l 1*)
by *simp*
have *102*: ⋀*j*. *j* ≤ *nlength* (*ndropn 1 l*) ⟶
(*nnth l 1*) ≤ (*nnth l* (*Suc j*))
by (*simp add*: *1 nidx-less-eq*)
have *103*: ⋀*j*. *j*< *nlength* (*ndropn 1 l*) ⟶
(*nnth l* (*Suc j*)) − (*nnth l 1*) <
(*nnth l* (*Suc* (*Suc j*))) − (*nnth l 1*)
using *1 nidx-expand*[*of l*]
by (*metis 102 4 diff-less-mono eSuc-enat ileI1 iless-Suc-eq order-less-imp-le*)
have *11*: *nidx* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*))
using *103 101 nidx-expand*[*of* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*))]
using *Suc-ile-eq* **by** *force*
have *12*: *nlength* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*)) = *n*
using *4* **by** *auto*
have *13*: (*enat* (*nlast* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*))))
=
(*nlength* (*ndropn* (*nnth l 1*) *σ*))
by (*metis 1 10 idiff-enat-enat ndropn-nlength nfinite-ndropn nlast-nmap*)
have *14*: (*nsubn σ 0* (*nnth l 1*) ⊨ *f*)
by (*metis 1 One-nat-def enat-ord-simps*(*2*) *zero-less-Suc*)
have *15*: (∀ (*i*::*nat*). *i* < *nlength* (*ndropn 1 l*) ⟶
(*nsubn σ* (*nnth* (*ndropn 1 l*) *i*) (*nnth* (*ndropn 1 l*) (*Suc i*)) ⊨ *f*))
using *1 3 eSuc-enat ileI1 iless-Suc-eq ndropn-nnth nlength-NCons plus-1-eq-Suc* **by** *metis*

have *16*: (∀ (*i*::*nat*). *i*< *nlength* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*)) ⟶

```
        (nsubn σ ((nnth (nmap (λx. x−  (nnth l 1)) (ndropn 1 l)) i) + (nnth l 1))
                 ((nnth (nmap (λx. x−  (nnth l 1)) (ndropn 1 l)) (Suc i)) + (nnth l 1))⊨ f))
  using 102 12 15 4 by force
 have 17: (∀ (i::nat). i< nlength (nmap (λx. x−  (nnth l 1)) (ndropn 1 l)) ⟶
            ((nsubn (ndropn (nnth l 1) σ)
                 (nnth (nmap (λx. x−  (nnth l 1)) (ndropn 1 l)) i)
                 (nnth (nmap (λx. x−  (nnth l 1)) (ndropn 1 l)) (Suc i))) ⊨ f))
  by (metis 11 16 eSuc-enat ileI1 nidx-expand nsubn-ndropn)
 have 18: nfinite (nmap (λx. x−  (nnth l 1)) (ndropn 1 l))
   using 12 nlength-eq-enat-nfiniteD by blast
 have 19: nfinite (ndropn (nnth l 1) σ)
   using 1 nfinite-ndropn-a by blast
 have 20: (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
                 (enat (nlast ls)) = (nlength (ndropn (nnth l 1) σ)) ∧ nfinite(ndropn (nnth l 1) σ) ∧
              (∀ (i::nat). i< (nlength ls) ⟶
                       ((nsubn (ndropn (nnth l 1) σ) (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
              ))
  by (metis 11 12 13  17 18 19 71 diff-self-eq-0 enat-le-plus-same(1) gen-nlength-def
     nlength-code nnth-nmap)
 show ?thesis
 by (metis 1 2 20 71 8 One-nat-def Suc-ile-eq finite-nidx-bounded-nlast zero-enat-def
    zero-order(1))
qed

lemma aschopstar-wpower-chain-b:
assumes (∃ k. 0 ≤k ∧ k ≤ nlength σ ∧ 0< k ∧
            (nsubn σ 0 k  ⊨ f) ∧
            (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
                 (enat (nlast ls)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
              (∀ (i::nat). i< (nlength ls) ⟶
                       ((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
              ))
        )
shows (∃ (l:: nat nellist).
             (nlength l) = (Suc n) ∧ nfinite l ∧ nidx l ∧ (nnth l 0) = 0 ∧
              (enat (nlast l)) = (nlength σ) ∧ nfinite σ ∧
                     (∀ (i::nat). i< (nlength l) ⟶
                        ((nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ f)
                     )
            )
proof −
 obtain k where 1: 0 ≤k ∧ k ≤ nlength σ ∧ k > 0 ∧
            (nsubn σ 0 k  ⊨ f) ∧
            (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
                 (enat (nlast ls)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
              (∀ (i::nat). i< (nlength ls) ⟶
                       ((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
              ))
 using assms by auto
 have 2: (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
```

$(enat\ (nlast\ ls)) = (nlength\ (ndropn\ k\ \sigma)) \wedge nfinite(ndropn\ k\ \sigma) \wedge$
$(\forall (i::nat).\ \ i< (nlength\ ls) \longrightarrow$
$((nsubn\ (ndropn\ k\ \sigma)\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ ) \models f)$
$))$
using *1* **by** *auto*
obtain *ls* **where** *3*: $nfinite\ ls \wedge (nlength\ ls) = n \wedge nidx\ ls \wedge (nnth\ ls\ 0) = 0 \wedge$
$(enat\ (nlast\ ls)) = (nlength\ (ndropn\ k\ \sigma)) \wedge nfinite(ndropn\ k\ \sigma) \wedge$
$(\forall (i::nat).\ \ i< (nlength\ ls) \longrightarrow$
$((nsubn\ (ndropn\ k\ \sigma)\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ ) \models f)$
$)$
using *2* **by** *auto*
have *4*: $nidx\ \ (nmap\ (\lambda x.\ x+\ k)\ ls)$
using *3*
by (*simp add*: *nidx-expand*)
have *41*: $\bigwedge j.\ j \leq nlength\ (nmap\ (\lambda x.\ x+\ k)\ ls) \longrightarrow$
$0 < (nnth\ \ (nmap\ (\lambda x.\ x+\ k)\ ls)\ j)$
by (*simp add*: *1*)
have *5*: $nidx\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ k)\ ls))$

using *3 4 nidx-expand*[*of ls*] *nidx-expand*[*of* $(nmap\ (\lambda x.\ x+\ k)\ ls)$]
nidx-expand[*of* $(NCons\ 0\ (nmap\ (\lambda x.\ x+\ k)\ ls))$]
by (*metis* (*no-types*, *lifting*) *1 Suc-ile-eq diff-zero iless-Suc-eq le-add-diff-inverse lessI*
less-Suc-eq-0-disj nlength-NCons nlength-nmap nnth-0 nnth-Suc-NCons nnth-nmap)
have *6*: $(nlength\ ((NCons\ 0\ (nmap\ (\lambda x.\ x+\ k)\ ls)))) = (Suc\ n)$
by (*simp add*: *3 eSuc-enat*)
have *7*: $(enat\ (nlast\ ((NCons\ 0\ (nmap\ (\lambda x.\ x+\ k)\ ls))))) = (nlength\ \sigma)$
by (*metis 1 3 add.commute enat.distinct*(*2*) *enat-add-sub-same less-eqE ndropn-nlength*
nlast-NCons nlast-nmap plus-enat-simps(*1*))
have *8*: $(nsubn\ \sigma\ 0\ k\ \models f)$
using *1* **by** *auto*

have *9*: $(\forall (i::nat).\ \ i< (nlength\ ls) \longrightarrow$
$((nsubn\ (ndropn\ k\ \sigma)\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ ) \models f)$
$)$
using *3* **by** *auto*
have *10*: $(\forall (i::nat).\ i< (nlength\ ls) \longrightarrow$
$((nsubn\ \sigma\ ((nnth\ ls\ (i))+k)\ ((nnth\ ls\ ((i)+1))+k)\ ) \models f)$
$)$
by (*metis 3 Suc-ile-eq add.commute add.right-neutral nidx-less nsubn-ndropn plus-1-eq-Suc*)

have *11*: $(\forall\ (i::nat).\ i < nlength\ ((((\ (nmap\ (\lambda x.\ x+\ k)\ ls)))) \longrightarrow$
$(nsubn\ \sigma\ (nnth\ ((((\ (nmap\ (\lambda x.\ x+\ k)\ ls)))))\ i)\ (nnth\ ((((\ (nmap\ (\lambda x.\ x+\ k)\ ls))))\ (Suc\ i)) \models f))$
by (*metis 10 add.commute eSuc-enat ileI1 nlength-nmap nnth-nmap order-less-imp-le plus-1-eq-Suc*)
have *12*: $(\forall i.\ (0<i \wedge i< 1+\ (nlength\ (nmap\ (\lambda x.\ x+\ k)\ ls))) \longrightarrow$
$((nsubn\ \sigma\ ((nnth\ (nmap\ (\lambda x.\ x+\ k)\ ls)\ (i-1)))$
$((nnth\ (nmap\ (\lambda x.\ x+\ k)\ ls)\ ((i))))\ ) \models f)$
$)$
using *11*
by (*metis 3 Suc-diff-1 Suc-ile-eq eSuc-enat iless-Suc-eq nlength-nmap one-enat-def plus-1-eq-Suc*

```
    plus-enat-simps(1))
 have 13: (∀ (i::nat). i < nlength ((NCons 0 ( (nmap (λx. x+ k) ls)))) ⟶
           (nsubn σ (nnth (((NCons 0 (nmap (λx. x+ k) ls)))) i)
                    (nnth (((NCons 0 (nmap (λx. x+ k) ls)))) (Suc i)) ⊨ f))
   using 12
   using 1 11 3 6 less-Suc-eq-0-disj by auto

 have 14: (nnth (NCons 0 (nmap (λx. x+ k) ls)) 0) = 0
  by simp
 have 15: (nnth (NCons 0 (nmap (λx. x+ k) ls)) (1)) = k
  using 4 by (simp add: 3)
 have 16: (nsubn σ (nnth (NCons 0 (nmap (λx. x+ k) ls)) 0) k  ⊨ f)
  by (simp add: 8)
 have 17: nfinite (NCons 0 (nmap (λx. x+ k) ls))
   using 6 nlength-eq-enat-nfiniteD by blast
 show ?thesis
 by (metis 13 3 5 6 7  nfinite-NCons nfinite-ndropn nfinite-nmap nnth-0)
qed

lemma chop-wpower-eqv-sem:
  ( (σ ⊨(∃ n. (wpower ((f ∧ more) ∧ finite) n)))) =
   ((σ ⊨ empty) ∨  ( (σ ⊨ ((f ∧ more) ∧ finite);(∃ n. (wpower ((f ∧ more) ∧ finite) n)))))
using wpowersem by fastforce

lemma aschopstar-eqv-wpower-chop-help:
   ( σ ⊨ wpower ((f ∧ more) ∧ finite) n) =
    (∃ (l:: nat nellist).
               (nlength l) = (n) ∧ nfinite l ∧ nidx l ∧ (nnth l 0) = 0 ∧
                (enat (nlast l)) = (nlength σ) ∧ nfinite σ ∧
                        (∀(i::nat). i< (nlength l) ⟶
                            ((nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ f)
                        )
              )

proof
 (induct n arbitrary: σ)
 case 0
 then show ?case
   by (auto simp add: empty-defs nidx-expand nnth-nlast zero-enat-def)
    (metis eSuc-enat enat-0-iff(1) iless-Suc-eq leD nfinite-conv-nlength-enat nlast-NNil nlength-NNil
     nnth-NNil zero-le)
 next
 case (Suc n)
 then show ?case
  proof −
   have (σ ⊨ wpower ((f ∧ more) ∧ finite) (Suc n)) =
        (σ ⊨ (((f ∧ more) ∧ finite);(wpower ((f ∧ more) ∧ finite) n)))
   by simp
   also have  ... =
            (∃ k. 0 ≤k ∧ k ≤ nlength (σ) ∧ k > 0 ∧
```

```
            (ntaken k (σ) ⊨ f) ∧
            (ndropn k (σ) ⊨ wpower ((f ∧ more) ∧ finite) n)
          )

  using enat-0-iff(1) le-zero-eq by (auto simp add: more-defs chop-defs finite-defs )
  also have ... =
        (∃ k. 0 ≤k ∧ k ≤nlength (σ) ∧ k > 0 ∧
            (nsubn σ 0 k  ⊨ f) ∧
            (ndropn  k (σ) ⊨ wpower ((f ∧ more) ∧ finite) n)
        )
    by (metis One-nat-def Suc-diff-1 Suc-diff-Suc diff-add ndropn-0 ntaken-ndropn)
  also have  ... =
        (∃ k. 0 ≤k ∧ k ≤ nlength (σ) ∧ k > 0 ∧
            (nsubn σ 0 k  ⊨ f) ∧
            (∃ l. nfinite l ∧ (nlength l) = n ∧ nidx l ∧ (nnth l 0) = 0 ∧
                  (enat (nlast l)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
              (∀(i::nat).  i< (nlength l) ⟶
                        ((nsubn (ndropn k σ) (nnth l i) (nnth l (Suc i)) ) ⊨ f)
              ))
          )
    using Suc.hyps by blast
   also have  ... =
        (∃ (l:: nat nellist).
            (nlength l) = (Suc n) ∧ nfinite l ∧ nidx l ∧ (nnth l 0) = 0 ∧
             (enat (nlast l)) = (nlength σ) ∧ nfinite σ ∧
                    (∀(i::nat). i< (nlength l) ⟶
                        ((nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ f)
                    )
           )

    using aschopstar-wpower-chain-a[of n σ f]
          aschopstar-wpower-chain-b [of σ f n] by auto
   finally show (σ ⊨ wpower ((f ∧ more) ∧ finite) (Suc n)) =
            (∃ (l:: nat nellist).
            (nlength l) = (Suc n) ∧ nfinite l ∧ nidx l ∧ (nnth l 0) = 0 ∧
             (enat (nlast l)) = (nlength σ) ∧ nfinite σ ∧

                    (∀(i::nat). i< (nlength l) ⟶
                        ((nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ f)
                    )
           ) .
  qed
qed

lemma aschopstar-eqv-power-chop:
  (σ ⊨ aschopstar f) = (  (σ ⊨ (∃ k. wpower ((f ∧ more) ∧ finite) k)))
using nfinite-conv-nlength-enat by (simp add: aschopstar-d-def aschopstar-eqv-wpower-chop-help )
blast
```

lemma *ASChopstarEqvSem*:
(σ ⊨ (*aschopstar f* = (*empty* ∨ ((*f* ∧ *more*) ∧ *finite*); (*aschopstar f*))))
proof −
have *1*: (σ ⊨ *aschopstar f*) = ((σ ⊨ (∃ *k*. *wpower* ((*f* ∧ *more*)∧ *finite*) *k*)))
using *aschopstar-eqv-power-chop* **by** *simp*
have *2*: ((σ ⊨ (∃ *k*. *wpower* ((*f* ∧ *more*)∧ *finite*) *k*))) =
((σ ⊨ *empty*) ∨ (σ ⊨ ((*f* ∧ *more*) ∧ *finite*);(∃ *n*. *wpower* ((*f* ∧ *more*)∧*finite*) *n*)))
using *chop-wpower-eqv-sem* **by** *simp*
have *3*: (σ ⊨ ((*f* ∧ *more*)∧*finite*);(∃ *n*. *wpower* ((*f* ∧ *more*)∧*finite*) *n*)) =
(∃ *n*. *n*≤*nlength* σ ∧ ((*ntaken n* σ) ⊨ (*f* ∧ *more*) ∧ *finite*) ∧
((*ndropn n* σ) ⊨ (∃ *x*. (*wpower* ((*f* ∧ *more*)∧*finite*) *x*))))
by (*simp add*: *chop-defs finite-defs*) *blast*
have *4*: (∃ *n*. *n*≤*nlength* σ ∧ ((*ntaken n* σ) ⊨ (*f* ∧ *more*) ∧ *finite*) ∧
((*ndropn n* σ) ⊨ (∃ *x*. (*wpower* ((*f* ∧ *more*)∧*finite*) *x*)))) =
(∃ *n*. *n*≤*nlength* σ ∧ ((*ntaken n* σ) ⊨ (*f* ∧ *more*)∧*finite*) ∧ ((*ndropn n* σ) ⊨ *aschopstar f*))
by (*simp add*: *aschopstar-eqv-power-chop*)
have *5*: (∃ *n*. *n*≤*nlength* σ ∧ ((*ntaken n* σ) ⊨ (*f* ∧ *more*)∧*finite*) ∧ ((*ndropn n* σ) ⊨ *aschopstar f*)) =
(σ ⊨ ((*f* ∧ *more*)∧*finite*); (*aschopstar f*))
by (*simp add*: *chop-defs finite-defs*) *blast*
have *6*: ((σ ⊨ *empty*) ∨ (σ ⊨ ((*f* ∧ *more*)∧*finite*);(∃ *n*. *wpower* ((*f* ∧ *more*)∧*finite*) *n*))) =
(σ ⊨ (*empty* ∨ ((*f* ∧ *more*)∧*finite*); (*aschopstar f*)))
using *3 4 5* **by** *auto*
show *?thesis* **using** *1 2 6* **by** *auto*
qed

lemma *ASChopstarEqvSChopstar*:
⊢ (*aschopstar f*) = (*schopstar f*)
by (*simp add*: *Valid-def schopstar-d-def aschopstar-eqv-power-chop fpowerstar-d-def fpower-d-def*)

lemma *len-defs* :
(*w* ⊨ *len n*) = (*nlength w* =*n*)
proof
(*simp add*: *len-d-def*)
show (*w* ⊨ (*wpower skip n*)) = (*nlength w* = *n*)
proof (*induct n arbitrary*:*w*)
case *0*
then show *?case* **by** (*simp add*: *empty-defs zero-enat-def*)
next
case (*Suc n*)
then show *?case*
by (*auto simp add*: *min-def len-d-def empty-defs chop-defs skip-defs finite-defs nlength-eq-enat-nfiniteD*)
(*metis One-nat-def enat.distinct*(*2*) *enat-add-sub-same le-iff-add plus-1-eq-Suc plus-enat-simps*(*1*))
qed
qed

lemma *PowerstarEqvSemhelp1*:
⊢ *empty*;(*empty* ∨ (*f* ∧ *inf*)) = (*empty* ∨ (*f* ∧ *inf*))

using *EmptyChopSem* **by** *blast*

lemma *PowerstarEqvSemhelp2*:
$\vdash (f \wedge inf) = (f \wedge inf);g$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *PowerstarEqvSemhelp3*:
$\vdash ((f \wedge inf);g \vee (f \wedge finite);g) = (f\ ;g)$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *WPowerstarEqvSem*:
$(\sigma \models (wpowerstar\ f) = \ (empty \vee \ f;(wpowerstar\ f)\ ))$
by (*metis intD wpowersem wpowerstar-d-def*)

lemma *FPowerstarEqvSem*:
$(\sigma \models (fpowerstar\ f) = \ (empty \vee \ (f \wedge finite);(fpowerstar\ f)\ ))$
by (*metis fpowersem fpowerstar-d-def intD*)

lemma *PowerstarEqvSem*:
$(\sigma \models (powerstar\ f) = \ (empty \vee \ f;(powerstar\ f)\ ))$
proof −
have *1*: $(\sigma \models (powerstar\ f)) =$
$(\ \sigma \models (\exists\ k.\ fpower\ f\ k);(empty \vee f \wedge inf))$
by (*simp add*: *powerstar-d-def fpowerstar-d-def*)
have *2*: $(\ \sigma \models (\exists\ k.\ fpower\ f\ k);(empty \vee f \wedge inf)) =$
$(\sigma \models (empty \vee (f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf))$
using *fpowersem* **by** (*metis inteq-reflection*)
have *3*: $(\sigma \models (empty \vee (f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf)) =$
$(\sigma \models empty;(empty \vee (f \wedge inf)) \vee$
$((f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf))$
by (*metis OrChopEqv inteq-reflection*)
have *4*: $(\sigma \models empty;(empty \vee (f \wedge inf)) \vee$
$((f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf)) =$
$(\sigma \models (empty \vee (f \wedge inf) \vee ((f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf)))$
using *PowerstarEqvSemhelp1*
by (*metis* (*mono-tags*, *lifting*) *inteq-reflection unl-lift2*)
have *5*: $(\sigma \models (empty \vee (f \wedge inf) \vee ((f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf))) =$
$(\sigma \models (empty \vee (f \wedge inf);((\exists\ k.\ (fpower\ f\ k));(empty \vee f \wedge inf)) \vee$
$((f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf)))$
using *PowerstarEqvSemhelp2*
by (*metis* (*mono-tags*, *lifting*) *inteq-reflection*)
have *51*: $(\sigma \models ((f \wedge finite);(\exists\ k.\ (fpower\ f\ k)));(empty \vee f \wedge inf) =$
$(f \wedge finite);((\exists\ k.\ (fpower\ f\ k));(empty \vee f \wedge inf)))$
by (*auto simp add*: *ChopAssocSemHelp1*)
have *6*: $(\sigma \models (empty \vee (f \wedge inf);((\exists\ k.\ (fpower\ f\ k));(empty \vee f \wedge inf)) \vee$

$$((f \wedge \mathit{finite});(\exists\ k.\ (\mathit{fpower}\ f\ k)));(\mathit{empty} \vee f \wedge \mathit{inf}))\ ) =$$
$$(\sigma \models (\mathit{empty} \vee (f \wedge \mathit{inf});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})) \vee$$
$$(f \wedge \mathit{finite});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf}))))$$

using *51* **by** *auto*
have *7* : $(\sigma \models (\mathit{empty} \vee (f \wedge \mathit{inf});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})) \vee$
$(f \wedge \mathit{finite});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})))) =$
$(\sigma \models (\mathit{empty} \vee f;((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf}))))$
using *PowerstarEqvSemhelp3* **by** *fastforce*
have *8*: $(\sigma \models (\mathit{empty} \vee f;((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})))) =$
$(\sigma \models (\mathit{empty} \vee f;(\mathit{powerstar}\ f)))$
by (*simp add*: *powerstar-d-def fpowerstar-d-def*)
from *1 2 3 4 5 6 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *wpowerchopsem*:
$\vdash (\exists\ k.\ \mathit{wpower}\ (f \wedge \mathit{more})\ k) =$
$(\ \mathit{empty} \vee (f \wedge \mathit{more});(\exists\ k.\ (\mathit{wpower}\ (f \wedge \mathit{more})\ k))$
$)$
by (*simp add*: *wpowersem*)

lemma *powerchopsem*:
$\vdash (\exists\ k.\ \mathit{fpower}\ (f \wedge \mathit{more})\ k) =$
$(\ \mathit{empty} \vee ((f \wedge \mathit{more}) \wedge \mathit{finite});(\exists\ k.\ (\mathit{fpower}\ (f \wedge \mathit{more})\ k))$
$)$
using *fpowersem* **by** *auto*

lemma *ChopstarEqvSem*:
$(\sigma \models f^\star = \ (\mathit{empty} \vee \ (f \wedge \mathit{more});\ f^\star)\ )$
by (*metis PowerstarEqvSem chopstar-d-def*)

lemma *SChopstarEqvSem*:
$(\sigma \models (\mathit{schopstar}\ f) = \ (\mathit{empty} \vee \ (f \wedge \mathit{more}) \frown (\mathit{schopstar}\ f)\ ))$
by (*metis FPowerstarEqvSem schop-d-def schopstar-d-def*)

lemma *ChopstarEqv* :
$\vdash\ f^\star = \ (\mathit{empty} \vee \ (f \wedge \mathit{more});\ f^\star)$
using *ChopstarEqvSem Valid-def* **by** *blast*

lemma *IStarIntros*:
$\vdash\ \mathit{empty} \vee f;(\mathit{istar}\ f) \longrightarrow (\mathit{istar}\ f)$
unfolding *Valid-def* **using** *istar-d.intros* **by** *fastforce*

lemma *IStarCases*:
$(w \models (\mathit{istar}\ F)) \implies$
$((w \models \mathit{empty} \vee (F;\mathit{istar}\ F)) \implies P) \implies P$

using *istar-d.cases*[*of F w P*] **by** *auto*

lemma *IStarEqvIStarSem*:
(*w* ⊨ (*istar f*)) = (*w* ⊨ *empty* ∨ *f*;(*istar f*))
using *IStarCases*[*of f*]
by (*metis istar-d.intros*(*1*) *istar-d.intros*(*2*) *unl-lift2*)

lemma *IStarEqvIStar*:
⊢ (*istar f*) = (*empty* ∨ *f*;(*istar f*))
using *IStarEqvIStarSem*[*of f*] **unfolding** *Valid-def* **by** *auto*

lemma *IStarInductSem*:
assumes (⋀*s*. (*s* ⊨ *empty*) ⟹ *P s*)
(⋀*s*. (*s* ⊨ (*F*;((*istar F*) ∧ *P*))) ⟹ *P s*)
shows (*w* ⊨ (*istar F*) ⟶ *P*)
using *assms istar-d.induct*[*of F w P*]
chop-defs[*of F LIFT* ((*istar F*) ∧ *P*)]
unfolding *chop-d-def* **by** (*metis intensional-rews*(*3*))

lemma *IStarInduct*:
assumes ⊢ *empty* ∨ *f*;((*istar f*) ∧ *g*) ⟶ *g*
shows ⊢ (*istar f*) ⟶ *g*
using *assms IStarInductSem*[*of g*] **unfolding** *Valid-def* **by** (*metis intensional-rews*(*3*))

lemma *IStarWeakInductSem*:
assumes (⋀*s*. (*s* ⊨ *empty*) ⟹ *P s*)
(⋀*s*. (*s* ⊨ (*F*;*P*)) ⟹ *P s*)
shows (*w* ⊨ (*istar F*) ⟶ *P*)
using *assms istar-d.induct*[*of F w P*] **using** *chop-defs*[*of F P*]
chop-defs[*of F LIFT* ((*istar F*) ∧ *P*)] **unfolding** *chop-d-def*
by (*metis intensional-rews*(*3*))

lemma *IStarWeakInduct*:
assumes ⊢ *empty* ∨ *f*; *g* ⟶ *g*
shows ⊢ (*istar f*) ⟶ *g*
using *assms IStarWeakInductSem*[*of g*] **unfolding** *Valid-def*
by (*metis intensional-rews*(*3*))

NFusecat definition of Chopstar

definition *bschopstar-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *bschopstar-d F* ≡
(λ*s*.
0< *nlength s* ⟶ (∃ *ss*.
nfinite ss ∧ *nfinite s* ∧ *s*= *nfusecat ss* ∧
nlastnfirst ss ∧
(∀ *i* ≤ *nlength ss*. *nfinite* (*nnth ss i*) ∧ *0* < *nlength* (*nnth ss i*)) ∧
(∀ *i* ≤ *nlength ss*. ((*nnth ss i*) ⊨ *F*))
)

)

syntax
-bschopstar-d :: *lift* ⇒ *lift* ((*bschopstar -*) [*85*] *85*)

syntax (*ASCII*)
-bschopstar-d :: *lift* ⇒ *lift* ((*bschopstar -*) [*85*] *85*)

translations
-bschopstar-d ⇌ *CONST bschopstar-d*

lemma *aschopstar-bschopstar-help1-zero*:
assumes
nlength σ = 0
shows ∃ *l*. *nfinite l* ∧ *nnth l 0 = 0* ∧ *nidx l* ∧ *nlast l* = (*the-enat*(*nlength σ*)) ∧
(∀ *i* ≤ *nlength l*. *enat* (*nnth l i*) ≤ *nlength* (*nfusecat ss*)) ∧
(∀ *i* < *nlength l*. *f* (*nsubn* (*nfusecat ss*) (*nnth l i*) (*nnth l* (*Suc i*))))
using *assms*
by (*metis enat-ord-simps*(*2*) *linorder-not-less nfinite-code*(*1*) *nidx-expand nlast-NNil nlength-NNil nnth-NNil not-iless0 the-enat-0 zero-enat-def zero-less-Suc*)

lemma *aschopstar-bschopstar-help1*:
assumes *nfinite ss*
nfinite σ
σ = nfusecat ss
nlastnfirst ss
∀ *i* ≤ *nlength ss*. *nfinite* (*nnth ss i*) ∧ *0*< *nlength* (*nnth ss i*)
∀ *i* ≤ *nlength ss*. *f* (*nnth ss i*)
0< *nlength σ*
shows ∃ *l*. *nfinite l* ∧ *nnth l 0 = 0* ∧ *nidx l* ∧ *nlast l* = (*the-enat*(*nlength σ*)) ∧
(∀ *i* ≤ *nlength l*. *enat* (*nnth l i*) ≤ *nlength* (*nfusecat ss*)) ∧
(∀ *i* < *nlength l*. *f* (*nsubn* (*nfusecat ss*) (*nnth l i*) (*nnth l* (*Suc i*))))
proof −
have *1*: *nfinite* (*addzero* (*lsum ss 0*))
using *assms* **by** (*simp add*: *addzero-def nfinite-lsum-conv*)
have *2*: *nnth* (*addzero* (*lsum ss 0*)) *0 =0*
using *assms lsum-nnth*[*of 0 ss 0*]
by (*metis lsum-addzero-nfirst nfirst-eq-nnth-zero*)
have *3*: *nidx* (*addzero* (*lsum ss 0*))
using *assms* **by** (*metis in-nset-conv-nnth lsum-addzero-nidx*)
have *31*: *nlength* (*nfusecat ss*) = *nlength σ*
using *assms* **by** *auto*
have *32*: *nlast* (*lsum ss 0*) = *nlength σ*
using *assms lsum-nlast*[*of ss 0*] *nfusecat-nlength-nfinite*[*of ss*]
sum-the-enat[*of ss the-enat* (*nlength ss*)]
by (*metis* (*no-types*, *lifting*) *add-0 in-nset-conv-nnth nfinite-conv-nlength-enat nle-le the-enat.simps*)
have *33*: *enat* (*nlast* (*addzero* (*lsum ss 0*))) = *nlength σ*
using *assms lsum-addzero-nlast*[*of ss*] *32* **by** *presburger*
have *4*: (∀ *i* ≤ *nlength* (*addzero* (*lsum ss 0*)). *enat* (*nnth* (*addzero* (*lsum ss 0*)) *i*) ≤ *nlength* (*nfusecat ss*))

using *assms finite-nidx-bounded-nlast*[*of* (*addzero* (*lsum ss 0*)) σ]
using *1 3 31 33* **by** *blast*
have *5*: $\bigwedge i$. *enat i* < *nlength* (*addzero* (*lsum ss 0*)) $\Longrightarrow$ *nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i* = *nnth ss i*
proof −
fix *i*
assume *ax*: *enat i* < *nlength* (*addzero* (*lsum ss 0*))
show *nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i* = *nnth ss i*
proof −
have *50*: *nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i* =
nsubn (*nfusecat ss*) (*nnth* ((*addzero* (*lsum ss 0*))) *i*) (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*))
using *nnth-nsubn-nellist*[*of i* (*addzero* (*lsum ss 0*)) (*nfusecat ss*)]
by (*simp add*: *Suc-ile-eq ax*)
have *51*: *nsubn* (*nfusecat ss*) (*nnth* ((*addzero* (*lsum ss 0*))) *i*) (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*)) =
ntaken (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*))
unfolding *nsubn-def1* **by** *simp*
have *52*: (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) =
nnth (*NCons 0* (*lsum ss 0*)) (*Suc i*) − *nnth* (*NCons 0* (*lsum ss 0*)) *i*
using *lsum-addzero-nnth*[*of i ss*] *lsum-addzero-nnth*[*of Suc i ss*]
using *ax* **by** (*metis addzero-def not-less-zero*)
have *53*: *nnth* (*NCons 0* (*lsum ss 0*)) (*Suc i*) − *nnth* (*NCons 0* (*lsum ss 0*)) *i* =
nnth (*lsum ss 0*) *i* − (*if i* = *0 then 0 else nnth* (*lsum ss 0*) (*i*−*1*))
using *assms* **apply** *simp*
by (*metis Suc-pred nnth-Suc-NCons*)
have *530*: *all-nfinite ss*
using *all-nfinite-nnth-a assms*(*5*) **by** *blast*
have *531*: *enat i* ≤ *nlength ss*
using *ax* **by** (*metis addzero-def iless-Suc-eq lsum-nlength nlength-NCons order-less-imp-le*)
have *54*: *nnth* (*lsum ss 0*) *i* = ($\sum$ *k*::*nat*= *0..i*. (*the-enat* (*nlength*(*nnth ss k*))))
using *assms lsum-nnth*[*of i ss 0*] **by** (*simp add*: *530 531*)
have *55*: *i*>*0* $\Longrightarrow$ *nnth* (*lsum ss 0*) (*i*−*1*) = ($\sum$ *k*::*nat*= *0..*(*i*−*1*). (*the-enat* (*nlength*(*nnth ss k*))))
using *assms lsum-nnth*[*of i*−*1 ss 0*]
using *530 531* **by** (*metis One-nat-def Suc-ile-eq Suc-pred add-0 order-less-imp-le*)
have *56*: *i*>*0* $\Longrightarrow$
($\sum$ *k*::*nat*= *0..i*. (*the-enat* (*nlength*(*nnth ss k*)))) −
($\sum$ *k*::*nat*= *0..*(*i*−*1*). (*the-enat* (*nlength*(*nnth ss k*)))) = (*the-enat*(*nlength*(*nnth ss i*)))
by (*metis Suc-diff-1 add-diff-cancel-left*′ *sum.atLeast0-atMost-Suc*)
have *57*: *i*=*0* $\Longrightarrow$ *nnth* (*lsum ss 0*) *i* − (*if i* = *0 then 0 else nnth* (*lsum ss 0*) (*i*−*1*)) =
(*the-enat*(*nlength*(*nnth ss i*)))
by (*metis* (*full-types*) *add-0 diff-zero lsum-nfirst nfirst-eq-nnth-zero*)
have *58*: *nnth* (*lsum ss 0*) *i* − (*if i* = *0 then 0 else nnth* (*lsum ss 0*) (*i*−*1*)) =
(*the-enat*(*nlength*(*nnth ss i*)))
using *54 55 56* **by** *auto*
have *59*: (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) =
(*the-enat*(*nlength*(*nnth ss i*)))
using *52 53 58* **by** *presburger*
have *60*: *ntaken* (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*)) =
ntaken (*the-enat*(*nlength*(*nnth ss i*))) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*))

by (*simp add*: *59*)
have *61*: (*nnth* (*addzero* (*lsum ss 0*)) *i*) =
(*the-enat* (*if i =0 then 0 else* (∑ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*))))))
using *lsum-addzero-nnth*[*of i ss*] *2 55 assms* **apply** *simp*
using *ax* **by** (*metis Suc-pred addzero-def nnth-Suc-NCons not-less-zero*)
have *62*: (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*)) =
(*ndropn* (*the-enat* (*if i =0 then 0 else* (∑ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*)))))) (*nfusecat ss*))
using *61* **by** *auto*
have *620*: *nlength* (*lsum ss 0*) = *nlength ss*
by (*simp add*: *lsum-nlength*)
have *621*: *nlength* (*addzero* (*lsum ss 0*)) = *nlength ss* + *1*
using *lsum-addzero-nlength*[*of ss*]
using *assms*(*5*) **by** (*metis add-0 i0-less less-le-not-le nfirst-eq-nnth-zero nle-le zero-enat-def*)
have *622*: *enat i* ≤ *nlength ss*
by (*simp add*: *531*)
have *623*: *enat i* < *nlength ss* ⟹
nfusecat (*ndropn i ss*) = *ndropn* (*the-enat* (*if i* = *0 then 0 else* ∑*j* = *0*..*i* − *1*. *nlength* (*nnth ss j*))) (*nfusecat ss*)
using *nfusecat-ndropn*[*of i ss*] *530 assms*(*4*) **by** *blast*
have *624*: ⋀*j*. *j* ≤ *i*−*1* ⟹ *nlength* (*nnth ss j*) = (*the-enat*(*nlength*(*nnth ss j*)))
using *622 assms*(*2*,*3*,*5*)
by (*metis* (*lifting*) *diff-le-self enat-le-plus-same*(*1*) *enat-ord-simps*(*1*) *enat-the-enat infinity-ileE nfinite-conv-nlength-enat*
order-trans)
have *625*: *enat i* < *nlength ss* ∧ *i*>*0*⟹
(∑*j* = *0*..*i* − *1*. *nlength* (*nnth ss j*)) = (∑ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*))))
by (*metis* (*no-types*, *lifting*) *530 One-nat-def Suc-ile-eq Suc-pred dual-order.strict-iff-order enat-the-enat sum.cong sum-finite*
sum-the-enat)
have *63*: *enat i* < *nlength ss* ⟹
nfusecat (*ndropn i ss*) = (*ndropn* (*the-enat* (*if i =0 then 0 else* (∑ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*)))))) (*nfusecat ss*))
using *2 623 625* **by** *auto*
have *64*: *enat i* < *nlength ss* ⟹ *ntaken* (*the-enat*(*nlength*(*nnth ss i*))) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*)) =
ntaken (*the-enat*(*nlength*(*nnth ss i*))) (*nfusecat* (*ndropn i ss*))
using *62 63* **by** *presburger*
have *65*: ⋀*n*. (∑*ia* = *0*..*n*. (*the-enat* (*nlength* (*nnth* (*ndropn i ss*) *ia*)))) =
(∑*ia* = *0*..*n*. (*the-enat* (*nlength* (*nnth ss* (*i*+ *ia*)))))
using *ndropn-nnth* **by** *auto*
have *66*: *enat i* < *nlength ss* ⟹ *nlength* (*ntaken* (*the-enat*(*nlength*(*nnth ss i*))) (*nfusecat* (*ndropn i ss*))) = (*the-enat*(*nlength*(*nnth ss i*)))
using *ntaken-nlength*[*of* (*the-enat*(*nlength*(*nnth ss i*))) (*nfusecat* (*ndropn i ss*))]
ndropn-nlength[*of* (*the-enat* (*if i =0 then 0 else* (∑ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*)))))) (*nfusecat ss*)]
63 4 59 61
by (*metis ax eSuc-enat enat-minus-mono1 idiff-enat-enat ileI1 min.orderE*)
have *67*: ⋀*j*. *enat j* ≤ (*the-enat*(*nlength*(*nnth ss i*))) ⟹
nnth (*ntaken* (*the-enat*(*nlength*(*nnth ss i*))) (*nfusecat* (*ndropn i ss*))) *j* =

nnth (nfusecat (ndropn i ss)) j
using *ntaken-nnth* **by** (*metis enat-ord-simps(1) min.orderE*)
have *68*: $\bigwedge j$. *enat i < nlength ss* $\Longrightarrow$ *enat j* $\leq$ (*the-enat*(*nlength*(*nnth ss i*))) $\Longrightarrow$
nnth (nfusecat (ndropn i ss)) j =
nnth (ndropn (the-enat(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1) . (the-enat*(*nlength*(*nnth ss i*)))))) *(nfusecat ss)) j*
using *63* **by** *force*
have *69*: $\bigwedge j$. *enat j* $\leq$ (*the-enat*(*nlength*(*nnth ss i*))) $\Longrightarrow$
nnth (ndropn (the-enat(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1) . (the-enat*(*nlength*(*nnth ss i*)))))) *(nfusecat ss)) j =*
nnth (nfusecat ss) ((the-enat(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1) . (the-enat*(*nlength*(*nnth ss i*))))))+*j*)
using *ndropn-nnth* **by** *simp*
have *70*: $\bigwedge j$. *enat j* $\leq$ (*the-enat*(*nlength*(*nnth ss i*))) $\Longrightarrow$
nnth (nfusecat ss) ((the-enat(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1) . (the-enat*(*nlength*(*nnth ss i*))))))+*j*) =
nnth (nfusecat ss) (nnth (addzero (lsum ss 0)) i + j)
by (*simp add*: *61*)
have *71*: *nlastnfirst ss* $\wedge$ *all-gr-zero ss* $\wedge$ *all-nfinite ss*
by (*metis assms*(*4*,*5*) *in-nset-conv-nnth*)
have *72*: $\bigwedge j$. *enat j* $\leq$ *nlength (nnth ss i)* $\Longrightarrow$ *nnth (nfusecat ss) (nnth (addzero (lsum ss 0)) i + j) = nnth (nnth ss i) j*
using *71* **using** *622 lsum-nfusecat-nnth-lsum-nnth* **by** *blast*
have *720* :*ntaken (nnth (addzero (lsum ss 0)) (Suc i) − nnth (addzero (lsum ss 0)) i)*
(ndropn (nnth (addzero (lsum ss 0)) i) (nfusecat ss)) =
ntaken (the-enat (nlength (nnth ss i)))
(ndropn (nnth (addzero (lsum ss 0)) i) (nfusecat ss))
using *60* **by** *blast*
have *721*: *nlength (ndropn (nnth (addzero (lsum ss 0)) i) (nfusecat ss)) = nlength σ − (nnth (addzero (lsum ss 0)) i)*
using *ndropn-nlength*[*of*] **by** (*simp add*: *assms*(*3*))
have *722*: *nfinite (nfirst ss)*
by (*metis assms*(*5*) *i0-less nfirst-eq-nnth-zero order-le-less zero-enat-def*)
have *723*: (*nlength ss = 0* $\wedge$ *nlength (nfirst ss) = 0* $\longrightarrow$ *nnth (addzero (lsum ss 0)) i = nnth (lsum ss 0) i*) $\wedge$
(*nlength ss = 0* $\wedge$ *0 < nlength (nfirst ss)* $\longrightarrow$ *nnth (addzero (lsum ss 0)) i = nnth (NCons 0 (lsum ss 0)) i*) $\wedge$
(*0 < nlength ss* $\longrightarrow$ *nnth (addzero (lsum ss 0)) i = nnth (NCons 0 (lsum ss 0)) i*)
using *722 ax lsum-addzero-nnth order-less-le* **by** *blast*
have *724*: *0 < nlength (nfirst ss)*
by (*metis assms*(*5*) *i0-lb nfirst-eq-nnth-zero zero-enat-def*)
have *725*: *nnth (addzero (lsum ss 0)) i = nnth (NCons 0 (lsum ss 0)) i*
using *723 724 i0-less* **by** *blast*
have *726*: *i = 0* $\Longrightarrow$ *nnth (NCons 0 (lsum ss 0)) i =0*
by *auto*
have *727*: *0 < i* $\Longrightarrow$ *nnth (NCons 0 (lsum ss 0)) i* = ($\sum$ *k*::*nat*= *0..i −1.* (*the-enat* (*nlength*(*nnth ss k*))))
using *61 725* **by** *auto*
have *728*: *i = 0* $\Longrightarrow$ (*the-enat* (*nlength* (*nnth ss i*))) $\leq$ *nlength σ* − (*nnth* (*addzero* (*lsum ss 0*)) *i*)
by (*metis 2 4 59 Suc-ile-eq assms*(*3*) *ax diff-zero idiff-enat-0-right*)

have *729*: $0 < i \Longrightarrow$ (*the-enat* (*nlength* (*nnth ss i*))) + ($\sum$ *k*::*nat*= *0..i −1.* (*the-enat* (*nlength*(*nnth ss k*)))) =
($\sum$ *k*::*nat*= *0..i* . (*the-enat* (*nlength*(*nnth ss k*))))
by (*metis* (*no-types*, *lifting*) *Suc-pred′ add.commute sum.atLeast0-atMost-Suc*)
have *730*: ($\sum$ *k*::*nat*= *0..i* . (*the-enat* (*nlength*(*nnth ss k*)))) ≤ *nlength σ*
by (*metis 4 54 722 724 assms*(*3*) *ax eSuc-enat i0-less ileI1 lsum-addzero-nnth nnth-Suc-NCons*)
have *731*: $0 < i \Longrightarrow$ ($\sum$ *k*::*nat*= *0..i −1.* (*the-enat* (*nlength*(*nnth ss k*)))) ≤ *nlength σ*
by (*metis 4 725 727 assms*(*3*) *ax order-less-le*)
have *732*: $0 < i \Longrightarrow$
(*the-enat* (*nlength* (*nnth ss i*))) ≤ *nlength σ* − ($\sum$ *k*::*nat*= *0..i −1.* (*the-enat* (*nlength*(*nnth ss k*)))) ⟷
($\sum$ *k*::*nat*= *0..i* . (*the-enat* (*nlength*(*nnth ss k*)))) ≤ *nlength σ*
by (*metis 56 730 enat-minus-mono1 idiff-enat-enat*)
have *733*: $0 < i \Longrightarrow$ (*the-enat* (*nlength* (*nnth ss i*))) ≤ *nlength σ* − ($\sum$ *k*::*nat*= *0..i −1.* (*the-enat* (*nlength*(*nnth ss k*))))
using *732 730* **by** *blast*
have *734*: (*the-enat* (*nlength* (*nnth ss i*))) ≤ *nlength σ* − (*nnth* (*addzero* (*lsum ss 0*)) *i*)
using *733 728 61* **by** *auto*
have *735*: *nlength* (*ntaken* (*the-enat* (*nlength* (*nnth ss i*))) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*))) =
(*the-enat* (*nlength* (*nnth ss i*)))
using *ntaken-nlength*[*of* (*the-enat* (*nlength* (*nnth ss i*)))] *734 721* **by** *simp*
have *73*: *nlength* (*nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i*) = *nlength* (*nnth ss i*)
using *735* **by** (*metis 50 622 720 assms*(*5*) *nfinite-nlength-enat nsubn-def1 the-enat.simps*)
have *74*: $\bigwedge$*ia. enat ia* ≤ *nlength* (*nnth ss i*) $\Longrightarrow$
nnth (*ntaken* (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*)
(*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*))) *ia* =
nnth (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*))
(*min ia* (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*))
using *ntaken-nnth*[*of* (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*)
(*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*)) -]
by *presburger*
have *75*: ($\bigwedge$*ia. enat ia* ≤ *nlength* (*nnth ss i*) $\Longrightarrow$
nnth (*nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i*) *ia* = *nnth* (*nnth ss i*) *ia*)
using *50 51 61 69 72 720 73 735 74*
using *59* **by** (*metis enat-ord-simps*(*1*) *min.orderE*)
show *?thesis* **using** *72 73 75 nellist-eq-nnth-eq*[*of nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i nnth ss i*]
by (*simp*)
qed
qed
have *5*: (∀ *i*< *nlength* (*addzero* (*lsum ss 0*)). *f* (*nsubn* (*nfusecat ss*) (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*))))
using *assms*
by (*metis 5 addzero-def eSuc-enat ileI1 iless-Suc-eq lsum-nlength nlength-NCons nnth-nsubn-nellist not-less-zero*)
have *6*: *nlast* (*addzero* (*lsum ss 0*)) = *the-enat* (*nlength σ*)
using *33* **by** (*metis the-enat.simps*)

show *?thesis*

using *1 2 3 4 5 6*
by *blast*
qed

lemma *aschopstar-bschopstar-help2*:
assumes *nfinite l*
nfinite σ
nnth l 0 = 0
nidx l
nlast l = (*the-enat*(*nlength σ*))
∀ *i* ≤ *nlength l*. *enat* (*nnth l i*) ≤ *nlength σ*
∀ *i*<*nlength l*. *f* (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)))
shows*0*< *nlength σ* ⟶
(∃ *ss*. *nfinite ss* ∧ *nfinite σ* ∧ *σ* = *nfusecat ss* ∧ *nlastnfirst ss* ∧
(∀ *i*≤*nlength ss*. *nfinite* (*nnth ss i*) ∧ *0*< *nlength* (*nnth ss i*)) ∧
(∀ *i*≤*nlength ss*. *f* (*nnth ss i*)))
proof −
have *1*: *nfinite* (*nfusecat* (*nsubn-nellist σ l*))
using *assms* **by** (*simp add*: *nfusecat-nsubn-nellist-finite*)
have *2*: *σ* = *nfusecat* (*nsubn-nellist σ l*)
using *assms* **by** (*simp add*: *nfirst-eq-nnth-zero nfusecat-nsubn-nellist-finite-var*)
have *3*: *nlastnfirst* (*nsubn-nellist σ l*)
using *assms 1 2 nlastnfirst-nsubn-nellist-finite*
by (*metis dual-order.refl i0-less nfinite-conv-nlength-enat nlastnfirst-NNil nsubn-nellist-def the-enat.simps*)
have *40*: *0*< *nlength σ* ⟶ *0* < *nlength l*
using *assms* **by** (*metis i0-less linorder-not-less ndropn-eq-NNil ndropn-nlast the-enat-0 zero-enat-def*)
have *41*: *0*< *nlength σ* ⟶ (∀ *i*≤ *nlength* (*nsubn-nellist σ l*). *nfinite* (*nnth* (*nsubn-nellist σ l*) *i*))
using *assms 40* **by** (*metis Suc-ile-eq iless-Suc-eq nellist-split-2-first nfinite-nnth-nsubn-nellist nlength-NCons nlength-nsubn-nellist*)
have *42*: *0*< *nlength σ* ⟶ (∀ *i*≤ *nlength* (*nsubn-nellist σ l*). *0* < *nlength* (*nnth* (*nsubn-nellist σ l*) *i*))
using *40 assms*
by (*metis Suc-ile-eq iless-Suc-eq nellist-split-2-first nidx-expand nlength-NCons nlength-nsubn-nellist nnth-nsubn-nellist*
nsubn-nlength-gr-one)
have *4*: *0*< *nlength σ* ⟶ (∀ *i*≤ *nlength* (*nsubn-nellist σ l*). *nfinite* (*nnth* (*nsubn-nellist σ l*) *i*) ∧ *nlength* (*nnth* (*nsubn-nellist σ l*) *i*) ≠ *0*)
using *41 42* **by** *auto*

have *5*: *0*< *nlength σ* ⟶ (∀ *i*≤ *nlength* (*nsubn-nellist σ l*). *f* (*nnth* (*nsubn-nellist σ l*) *i*))
using *assms 40*
by (*metis Suc-ile-eq iless-Suc-eq nellist-split-2-first nlength-NCons nlength-nsubn-nellist nnth-nsubn-nellist*)
show *?thesis*
using *1 2 3 4 5 in-nset-conv-nnth nfusecat-nfinite not-gr-zero zero-enat-def* **by** *metis*
qed

lemma *aschopstar-bschopstar-sem*:
(*σ* ⊨ (*aschopstar f*)) = (*σ* ⊨ (*bschopstar f*))
proof −
have *1*: (*σ* ⊨ (*aschopstar f*)) ⟹ (*σ* ⊨ (*bschopstar f*))

unfolding *bschopstar-d-def aschopstar-d-def*
using *aschopstar-bschopstar-help2*[*of - σ f*]
by (*metis finite-nidx-bounded-nlast the-enat.simps*)
have *2*: (*σ* ⊨ (*bschopstar f*)) ⟹ (*σ* ⊨ (*aschopstar f*))
unfolding *bschopstar-d-def aschopstar-d-def*
using *aschopstar-bschopstar-help1*[*of - σ f*]
aschopstar-bschopstar-help1-zero[*of σ - f*]
apply (*auto simp add*: *nfinite-conv-nlength-enat zero-enat-def*)
apply (*metis is-NNil-nappend is-NNil-ndropn nappend-ntaken-ndropn ndropn-NNil nidx-expand nlast-NNil nlength-NNil nnth-NNil not-gr-zero not-iless0 zero-enat-def zero-le*)
by (*metis enat-ord-simps*(*1*) *gr-zeroI the-enat.simps zero-enat-def*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *aschopstar-eqv-bschopstar-sem*:
(*σ* ⊨ (*aschopstar f*) = (*bschopstar f*))
using *aschopstar-bschopstar-sem* **by** *auto*

lemma *aschopstar-eqv-bschopstar*:
⊢ (*aschopstar f*) = (*bschopstar f*)
using *aschopstar-eqv-bschopstar-sem* **unfolding** *Valid-def* **by** *auto*

3.5.3 Helper lemmas

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:
⊢ (*f* ∧ *empty*);(*f* ∧ *empty*) = (*f* ∧ *empty*)
proof −
have *1*: ⊢ (*f* ∧ *empty*);(*f* ∧ *empty*) ⟶ (*f* ∧ *empty*)
by (*metis ChopAndB ChopEmpty int-eq*)
have *2*: ⊢ (*f* ∧ *empty*) ⟶ (*f* ∧ *empty*);(*f* ∧ *empty*)
by (*auto simp add*: *Valid-def itl-defs zero-enat-def*)
(*metis iless-Suc-eq less-numeral-extra*(*1*) *ntake-0 ntake-all one-eSuc zero-enat-def*)
show *?thesis*
by (*simp add*: *1 2 int-iffI*)
qed

3.5.4 Properties of Chopstar and Chopplus

lemma *FPowerstardef*:
⊢ *fpowerstar f* = (∃ *n*. *fpower f n*)
by (*simp add*: *fpowerstar-d-def*)

lemma *Powerstardef*:
⊢ *powerstar f* = (*fpowerstar f*);(*empty* ∨ (*f* ∧ *inf*))
by (*simp add*: *fpowerstar-d-def powerstar-d-def*)

lemma *Chopstardef*:
⊢ *chopstar f* = *powerstar* (*f* ∧ *more*)
by (*simp add*: *chopstar-d-def*)

lemma *SChopstardef*:
$\vdash$ *schopstar f* = *fpowerstar* (*f* $\wedge$ *more*)
by (*simp add*: *schopstar-d-def*)

lemma *WPowerEqRule*:
assumes $\vdash$ *f* = *g*
shows $\vdash$ *wpower f n* = *wpower g n*
using *assms*
by (*metis int-eq int-simps*(*20*))

lemma *WPowerCommute*:
$\vdash$ (*f*) ;(*wpower f n*) = (*wpower f n*);(*f*)
proof
(*induct n*)
case *0*
then show *?case*
by (*metis ChopEmpty EmptyChop inteq-reflection wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis ChopAssoc inteq-reflection wpow-Suc*)
qed

lemma *FPowerCommute*:
$\vdash$ (*f* $\wedge$ *finite*) ;*fpower f n* = *fpower f n*;(*f* $\wedge$ *finite*)
unfolding *fpower-d-def* **using** *WPowerCommute*[*of LIFT* (*f* $\wedge$ *finite*)]
by *blast*

lemma *WPowerChopInductL*:
assumes $\vdash$ *g* $\vee$ *f*;*h* $\longrightarrow$ *h*
shows $\vdash$ (*wpower f n*);*g* $\longrightarrow$ *h*
using *assms*
proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChop*
by (*metis MP Prop12 int-eq int-iffD1 int-simps*(*33*) *wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis ChopAssoc Prop05 Prop11 RightChopImpChop lift-imp-trans wpow-Suc*)
qed

lemma *FPowerChopInductL*:
assumes $\vdash$ *g* $\vee$ *f*;*h* $\longrightarrow$ *h*
shows $\vdash$ (*fpower f n*);*g* $\longrightarrow$ *h*
unfolding *fpower-d-def* **using** *assms WPowerChopInductL*[*of g LIFT* (*f* $\wedge$ *finite*) *h n*]
using *AndChopA* **by** *fastforce*

lemma *FPowerChopInductFiniteL*:
assumes $\vdash g \vee (f \wedge \mathit{finite});h \longrightarrow h$
shows $\vdash (\mathit{fpower}\ f\ n);g \longrightarrow h$
unfolding *fpower-d-def* **using** *assms WPowerChopInductL*[*of g LIFT* $(f \wedge \mathit{finite})$ *h n*]
by *blast*

lemma *WPowerChopInductMoreL*:
assumes $\vdash g \vee (f \wedge \mathit{more});h \longrightarrow h$
shows $\vdash (\mathit{wpower}\ f\ n);g \longrightarrow h$
using *assms*
proof
(*induct n*)
case *0*
then show *?case*
by (*metis WPowerChopInductL wpow-0*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: $\vdash \mathit{wpower}\ f\ (\mathit{Suc}\ n);g = (f;\mathit{wpower}\ f\ n);g$
by *simp*
have *2*: $\vdash (f;\mathit{wpower}\ f\ n);g = f;((\mathit{wpower}\ f\ n);g)$
by (*meson ChopAssoc Prop11*)
have *3*: $\vdash f;((\mathit{wpower}\ f\ n);g) \longrightarrow f;h$
using *RightChopImpChop Suc.hyps Suc.prems* **by** *blast*
have *31*: $\vdash f = ((f \wedge \mathit{more}) \vee (f \wedge \mathit{empty}))$
unfolding *empty-d-def* **by** *fastforce*
have *4*: $\vdash f;h = ((\ (f \wedge \mathit{more}));\ h \vee ((f \wedge \mathit{empty}));h)$
using *31 OrChopEqvRule* **by** *blast*
have *5*: $\vdash (\ (f \wedge \mathit{more}));\ h \longrightarrow h$
by (*metis Prop03 Prop10 assms inteq-reflection lift-imp-trans*)
have *6*: $\vdash (\ (f \wedge \mathit{empty}));\ h \longrightarrow h$
by (*metis AndChopB EmptyChop inteq-reflection*)
from *5 6 4 3 2 1* **show** *?thesis*
by (*metis Prop02 inteq-reflection lift-imp-trans*)
qed
qed

lemma *FPowerChopInductFiniteMoreL*:
assumes $\vdash g \vee ((f \wedge \mathit{finite}) \wedge \mathit{more});h \longrightarrow h$
shows $\vdash (\mathit{fpower}\ f\ n);g \longrightarrow h$
unfolding *fpower-d-def* **using** *assms*
WPowerChopInductMoreL[*of g LIFT* $(f \wedge \mathit{finite})$ *h n*]
by *blast*

lemma *FPowerChopInductInfL*:
assumes $\vdash g \vee f;h \longrightarrow h$

shows $\vdash ((fpower\ f\ n);(f \wedge inf));g \longrightarrow h$
using *assms*
proof
(*induct n*)
case *0*
then show *?case*
by (*metis* (*no-types*, *lifting*) *AndInfChopEqvAndInf ChopAssoc FPowerChopInductFiniteL PowerstarEqvSemhelp3*
Prop03 Prop10 Prop12 inteq-reflection)
next
case (*Suc n*)
then show *?case*
proof −
have $\vdash(f \wedge finite);(fpower\ f\ n;((f \wedge inf);g)) = (fpower\ f\ (Suc\ n);(f \wedge inf));g$
by (*metis ChopAssoc fpower-d-def int-eq wpow-Suc*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *AndChopA ChopAndB ChopAssoc Prop03 Prop10 Suc assms int-eq lift-imp-trans*)

qed
qed

lemma *FChopInductInfMoreL*:
assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash ((fpower\ f\ n);((f \wedge more) \wedge inf));g \longrightarrow h$
using *FPowerChopInductInfL*
by (*metis AndMoreAndInfEqvAndInf assms inteq-reflection*)

lemma *WPowerChopInductR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;(wpower\ f\ n) \longrightarrow h$
using *assms*
proof
(*induct n*)
case *0*
then show *?case* **using** *ChopEmpty*
by (*metis MP Prop12 int-iffD2 int-simps*(*33*) *inteq-reflection wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis AndChopB ChopAssoc Prop03 Prop10 WPowerCommute inteq-reflection lift-imp-trans wpow-Suc*)
qed

lemma *FPowerChopInductR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;(fpower\ f\ n) \longrightarrow h$
unfolding *fpower-d-def* **using** *assms WPowerChopInductR*[*of g h LIFT* $(f \wedge finite)$ *n*]
using *ChopAndA* **by** *fastforce*

lemma *FpowerChopInductInfR*:

assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g;((fpower\ f\ n);(f \wedge inf)) \longrightarrow h$
using *assms*
by (*metis ChopAndA ChopAssoc FPowerChopInductR LeftChopImpChop Prop05 int-iffD1 lift-imp-trans*)

lemma *WPowerStarCommute*:
$\vdash f;(\exists\ n.\ wpower\ f\ n) = (\exists\ n.\ wpower\ f\ n);f$
proof −
have *1*: $\vdash f\ ;(\exists\ n.\ wpower\ f\ n) = (\exists\ n.\ f\ ;wpower\ f\ n)$
by (*metis ChopExist Prop11*)
have *2*: $\vdash (\exists\ n.\ f\ ;wpower\ f\ n) = (\exists\ n.\ (wpower\ f\ n);f\ )$
using *WPowerCommute* **by** (*metis ExEqvRule*)
have *3*: $\vdash (\exists\ n.\ (wpower\ f\ n);f) = (\exists\ n.\ (wpower\ f\ n));f$
by (*simp add*: *ExistChop*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FPowerStarCommute*:
$\vdash (f \wedge finite);(\exists\ n.\ fpower\ f\ n) = (\exists\ n.\ fpower\ f\ n);(f \wedge finite)$
unfolding *fpower-d-def* **using** *WPowerStarCommute*[*of LIFT* $(f \wedge finite)$]
by *blast*

lemma *WPowerSucAndEmptyEqvAndEmpty*:
$\vdash (wpower\ (f \wedge empty)\ (Suc\ n)) = (f \wedge empty)$
proof
(*induct n*)
case *0*
then show *?case*
by (*metis ChopEmpty wpow-0 wpow-Suc*)
next
case (*Suc n*)
then show *?case*
by (*metis AndEmptyChopAndEmptyEqvAndEmpty int-eq wpow-Suc*)
qed

lemma *FPowerSucAndEmptyEqvAndEmpty*:
$\vdash (fpower\ (f \wedge empty)\ (Suc\ n)) = (f \wedge empty)$
unfolding *fpower-d-def* **using** *WPowerSucAndEmptyEqvAndEmpty*[*of LIFT* $(f \wedge finite)\ n$]
WPowerEqRule[*of LIFT* $((f \wedge empty) \wedge finite)$ *LIFT* $((f \wedge finite) \wedge empty)\ (Suc\ n)$]
by (*meson EmptyImpFinite Prop01 Prop04 Prop05 Prop10 WPowerEqRule WPowerSucAndEmptyEqvAndEmpty*)

lemma *WPowerOr*:
$\vdash\ (wpower\ (f \vee g)\ (Suc\ n)) = (\ (f;wpower\ (f \vee g)\ n) \vee (g;wpower\ (f \vee g)\ n))$
by (*simp add*: *OrChopEqv*)

lemma *FPowerOr*:
$\vdash\ (fpower\ (f \vee g)\ (Suc\ n)) = (\ ((f \wedge finite);fpower\ (f \vee g)\ n) \vee$
$((g \wedge finite);fpower\ (f \vee g)\ n))$
by (*simp add*: *FiniteOr OrChopEqvRule fpower-d-def*)

lemma *WPowerEmptyOrMore*:
$\vdash$ (*wpower* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) (*Suc n*)) =
((*f* $\wedge$ *empty*);(*wpower* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) *n*) $\vee$
(*f* $\wedge$ *more*);(*wpower* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) *n*))
using *WPowerOr* **by** *blast*

lemma *FPowerEmptyOrMore*:
$\vdash$ (*fpower* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) (*Suc n*)) =
((*f* $\wedge$ *empty*);(*fpower* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) *n*) $\vee$
(*f* $\wedge$ *fmore*);(*fpower* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) *n*))
using *FPowerOr*[*of LIFT* (*f* $\wedge$ *empty*) *LIFT* (*f* $\wedge$ *more*) *n*]
by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore EmptyImpFinite Prop01 Prop05 Prop10 int-eq*)

lemma *WPowerstarInductL*:
assumes $\vdash$ *g* $\vee$ *f*;*h* $\longrightarrow$ *h*
shows $\vdash$ (*wpowerstar f*);*g* $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ (*wpowerstar f*);*g* = ((∃ *n*. *wpower* (*f*) *n*));*g*
by (*simp add*: *wpowerstar-d-def LeftChopEqvChop*)
have *2*: $\vdash$ (∃ *n*. *wpower* (*f*) *n*);*g* =
(∃ *n*. (*wpower* (*f*) *n*);*g*)
by (*metis ExistChop inteq-reflection*)
have *3*: $\bigwedge$ *n*. $\vdash$ (*wpower* (*f*) *n*);*g* $\longrightarrow$ *h*
using *WPowerChopInductL*[*of g LIFT*(*f*) *h*] *assms* **by** *auto*
have *4*: $\vdash$ (∃ *n*. ((*wpower* (*f*) *n*));*g*) $\longrightarrow$ *h*
by (*metis* (*mono-tags*, *lifting*) *3 Prop10 intI int-eq unl-Rex unl-lift2*)
from *1 2 4* **show** *?thesis*
by (*metis inteq-reflection*)
qed

lemma *FPowerstar-WPowerstar*:
$\vdash$ *fpowerstar f* = *wpowerstar* (*f* $\wedge$ *finite*)
unfolding *fpowerstar-d-def wpowerstar-d-def fpower-d-def* **by** *simp*

lemma *FPowerstarInductL*:
assumes $\vdash$ *g* $\vee$ (*f* $\wedge$ *finite*);*h* $\longrightarrow$ *h*
shows $\vdash$ (*fpowerstar f*);*g* $\longrightarrow$ *h*
using *assms WPowerstarInductL*[*of g LIFT* (*f* $\wedge$ *finite*) *h*]
FPowerstar-WPowerstar[*of f*] **by** (*metis int-eq*)

lemma *WPowerstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*wpowerstar f*) $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ *g*;(*wpowerstar f*) = *g*;(∃ *n*. *wpower f n*)
by (*simp add*: *wpowerstar-d-def*)
have *2*: $\vdash$ (*g*;(∃ *n*. *wpower f n*)) = (∃ *n*. *g*;(*wpower f n*))

by (*metis Prop04 ChopExist int-simps*(*31*))
have *3*: $\bigwedge n. \vdash g;(wpower\ f\ n) \longrightarrow h$
using *WPowerChopInductR assms* **by** *blast*
have *4*: $\vdash (\exists\ n.\ g;(wpower\ f\ n)) \longrightarrow h$
by (*metis* (*mono-tags*, *lifting*) *3 Prop10 intI int-eq unl-Rex unl-lift2*)
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *FPowerstarInductR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;(fpowerstar\ f) \longrightarrow h$
proof −
have *1*:$\vdash g \vee h;(f \wedge finite) \longrightarrow h$
using *assms* **using** *ChopAndA* **by** *fastforce*
show *?thesis*
using *1 WPowerstarInductR*[*of g h LIFT* $(f \wedge finite)$]
FPowerstar-WPowerstar[*of f*]
by (*metis inteq-reflection*)
qed

lemma *WPowerstarEqv* :
$\vdash (wpowerstar\ f) = (empty \vee f;(wpowerstar\ f))$
using *WPowerstarEqvSem* **by** *blast*

lemma *FPowerstarEqv* :
$\vdash (fpowerstar\ f) = (empty \vee (f \wedge finite);(fpowerstar\ f))$
by (*simp add*: *fpowersem fpowerstar-d-def*)

lemma *SChopstarEqv* :
$\vdash (schopstar\ f) = (empty \vee ((f \wedge more) \wedge finite);(schopstar\ f))$
by (*simp add*: *FPowerstarEqv schopstar-d-def*)

lemma *WPowerstar-more-absorb*:
$\vdash (wpowerstar\ (f \wedge more)) = (wpowerstar\ f)$
proof −
have *1*: $\vdash (wpowerstar\ (f \wedge more)) \longrightarrow (wpowerstar\ f)$
using *WPowerstarInductL*[*of LIFT empty LIFT f* $\wedge$ *more LIFT* $(wpowerstar\ f)$]
by (*metis AndChopA ChopEmpty Prop02 Prop05 WPowerChopInductL WPowerstarEqv int-iffD2 inteq-reflection wpow-0*)
have *2*: $\vdash empty \longrightarrow wpowerstar\ (f \wedge more)$
using *WPowerstarEqv*[*of LIFT f* $\wedge$ *more*] **by** *fastforce*
have *20*: $\vdash (f \wedge more);wpowerstar\ (f \wedge more) \longrightarrow wpowerstar\ (f \wedge more)$
by (*meson Prop03 WPowerstarEqv*)
have *21*: $\vdash (f \wedge empty);wpowerstar\ (f \wedge more) \longrightarrow wpowerstar\ (f \wedge more)$
by (*metis AndChopB EmptyChop int-eq*)
have *22*: $\vdash ((f \wedge more) \vee (f \wedge empty));wpowerstar\ (f \wedge more) =$
$((f \wedge more);wpowerstar\ (f \wedge more) \vee (f \wedge empty);wpowerstar\ (f \wedge more))$
by (*simp add*: *OrChopEqv*)
have *23*: $\vdash ((f \wedge more) \vee (f \wedge empty)) = f$

unfolding *empty-d-def* **by** *fastforce*
have *3*: $\vdash f;wpowerstar\ (f \wedge more) \longrightarrow wpowerstar\ (f \wedge more)$
by (*metis 20 21 22 23 Prop02 inteq-reflection*)
have *4*: $\vdash empty \vee f;wpowerstar\ (f \wedge more) \longrightarrow wpowerstar\ (f \wedge more)$
using *2 3* **by** *fastforce*
have *5*: $\vdash (wpowerstar\ f) \longrightarrow (wpowerstar\ (f \wedge more))$
using *4 WPowerstarInductL*[*of LIFT empty f LIFT* $(wpowerstar\ (f \wedge more))$]
by (*metis ChopEmpty inteq-reflection*)
show *?thesis* **using** *1 5* **by** *fastforce*
qed

lemma *FPowerstar-more-absorb*:
$\vdash (fpowerstar\ (f \wedge more)) = (fpowerstar\ f)$
proof −
have *1*: $\vdash ((f \wedge more) \wedge finite) = ((f \wedge finite) \wedge more)$
by *fastforce*
show *?thesis*
using *FPowerstar-WPowerstar*[*of f*] *FPowerstar-WPowerstar*[*of LIFT* $(f \wedge more)$]
WPowerstar-more-absorb[*of LIFT* $(f \wedge finite)$] *1*
by (*metis int-eq*)
qed

lemma *SChopstar-WPowerstar*:
$\vdash (schopstar\ f) = (wpowerstar\ (f \wedge finite))$
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb inteq-reflection schopstar-d-def*)

lemma *SChopstar-and-more*:
$\vdash (schopstar\ (f \wedge more)) = (schopstar\ f)$
by (*simp add*: *FPowerstar-more-absorb schopstar-d-def*)

lemma *IStarWPowerstar*:
$\vdash (istar\ f) = (wpowerstar\ f)$
proof −
have *1*: $\vdash (istar\ f) \longrightarrow (wpowerstar\ f)$
by (*metis IStarWeakInduct WPowerstarEqv int-iffD2*)
have *2*: $\vdash (wpowerstar\ f) \longrightarrow (istar\ f)$
using *WPowerstarInductL*[*of LIFT empty f LIFT istar f*]
by (*metis ChopEmpty IStarIntros inteq-reflection*)
show *?thesis*
by (*simp add*: *1 2 Prop11*)
qed

3.5.5 Kleene Algebra

lemma *WPowerstar-imp-empty*:
$\vdash empty \longrightarrow (wpowerstar\ f)$
using *WPowerstarEqv*[*of f*] **by** *fastforce*

lemma *SChopstar-imp-empty*:

$\vdash$ *empty* $\longrightarrow$ (*schopstar f*)
using *SChopstarEqv*[*of f*] **by** *fastforce*

lemma *WPowerstar-swap*:
$\vdash$ (*wpowerstar* ($f \vee g$)) = (*wpowerstar* ($g \vee f$))
proof −
have *1*: $\vdash$ ($f \vee g$) = ($g \vee f$)
by *fastforce*
show *?thesis*
by (*metis 1 WPowerstarEqv inteq-reflection*)
qed

lemma *SChopstar-swap*:
$\vdash$ (*schopstar* ($f \vee g$)) = (*schopstar* ($g \vee f$))
proof −
have *1*: $\vdash$ ($f \vee g$) = ($g \vee f$)
by *fastforce*
show *?thesis*
by (*metis 1 SChopstardef int-eq*)
qed

lemma *WPowerstar-1L*:
$\vdash$ *f*;(*wpowerstar f*) $\longrightarrow$ (*wpowerstar f*)
by (*meson Prop03 WPowerstarEqv*)

lemma *SChopstar-1L*:
$\vdash$ (*f*)⌢(*schopstar f*) $\longrightarrow$ (*schopstar f*)
by (*metis SChopstar-WPowerstar WPowerstar-1L inteq-reflection schop-d-def*)

lemma *SChopstarMore-1L*:
$\vdash$ ($f \wedge$ *more*)⌢(*schopstar f*) $\longrightarrow$ (*schopstar f*)
by (*meson AndSChopA SChopstar-1L lift-imp-trans*)

lemma *WPowerstar-trans-eq*:
$\vdash$ (*wpowerstar f*);(*wpowerstar f*) = (*wpowerstar f*)
proof −
have *1*: $\vdash$ (*wpowerstar f*) $\vee$ *f*;(*wpowerstar f*) $\longrightarrow$ (*wpowerstar f*)
by (*simp add*: *WPowerstar-1L*)
have *2*: $\vdash$ (*wpowerstar f*);(*wpowerstar f*) $\longrightarrow$ (*wpowerstar f*)
using *1 WPowerstarInductL* **by** *blast*
have *3*: $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*wpowerstar f*);(*wpowerstar f*)
by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *SChopstar-trans-eq*:
$\vdash$ (*schopstar f*);(*schopstar f*) = (*schopstar f*)
by (*metis SChopstar-WPowerstar WPowerstar-trans-eq inteq-reflection*)

lemma *WPowerstar-trans*:
$\vdash (wpowerstar\ f);(wpowerstar\ f) \longrightarrow (wpowerstar\ f)$
using *WPowerstar-trans-eq* **by** *fastforce*

lemma *SChopstar-trans*:
$\vdash (schopstar\ f);(schopstar\ f) \longrightarrow (schopstar\ f)$
using *SChopstar-trans-eq* **by** *fastforce*

lemma *WPowerstar-induct-lvar*:
assumes $\vdash f;g \longrightarrow g$
shows $\vdash (wpowerstar\ f);g \longrightarrow g$
using *assms*
by (*simp add*: *WPowerstarInductL*)

lemma *SChopstar-induct-lvar*:
assumes $\vdash (f)^\frown g \longrightarrow g$
shows $\vdash (schopstar\ f)^\frown g \longrightarrow g$
using *assms*
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-lvar inteq-reflection lift-imp-trans schop-d-def*)

lemma *SChopstarMore-induct-lvar*:
assumes $\vdash (f \wedge more)^\frown g \longrightarrow g$
shows $\vdash (schopstar\ f)^\frown g \longrightarrow g$
using *assms*
by (*metis FPowerstar-WPowerstar SChopstar-induct-lvar SChopstar-WPowerstar inteq-reflection schop-star-d-def*)

lemma *WPowerstar-inductL-var-equiv*:
$(\vdash (wpowerstar\ f);g \longrightarrow g) = (\vdash f;g \longrightarrow g)$
proof −
have *1*: $(\vdash f;g \longrightarrow g) \Longrightarrow (\vdash (wpowerstar\ f);g \longrightarrow g)$
by (*simp add*: *WPowerstar-induct-lvar*)
have *2*: $(\vdash (wpowerstar\ f);g \longrightarrow g) \Longrightarrow (\vdash f;g \longrightarrow g)$
by (*metis* (*no-types*, *lifting*) *AndChopB ChopAssoc EmptyChop Prop10 WPowerstar-1L WPowerstar-imp-empty int-eq lift-and-com*)
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *SChopstar-inductL-var-equiv*:
$(\vdash (schopstar\ f)^\frown g \longrightarrow g) = (\vdash (f)^\frown g \longrightarrow g)$
proof −
have *1*: $(\vdash (f)^\frown g \longrightarrow g) \Longrightarrow (\vdash (schopstar\ f)^\frown g \longrightarrow g)$
by (*simp add*: *SChopstar-induct-lvar*)
have *10*: $\vdash (schopstar\ f) \longrightarrow (empty \vee (f \wedge finite) \vee f^\frown(schopstar\ f))$
by (*metis AndSChopA FPowerstarEqv Prop05 Prop08 Prop11 schop-d-def schopstar-d-def*)
have *101*: $\vdash (f \wedge finite) \longrightarrow (schopstar\ f)$
by (*metis ChopEmpty RightChopImpChop SChopstar-1L SChopstar-imp-empty int-eq lift-imp-trans schop-d-def*)
have *11*: $\vdash (empty \vee (f \wedge finite) \vee f^\frown(schopstar\ f)) \longrightarrow (schopstar\ f)$
by (*meson 101 Prop02 SChopstar-1L SChopstar-imp-empty*)

have *12*: $\vdash (schopstar\ f) = (empty \vee (f \wedge finite) \vee f \frown (schopstar\ f))$
using *10 11 int-iffI* **by** *blast*
have *2*: $(\vdash (schopstar\ f) \frown g \longrightarrow g) \Longrightarrow (\vdash (f) \frown g \longrightarrow g)$
using *12*
by (*metis* (*no-types*, *opaque-lifting*) *EmptySChop LeftSChopImpSChop SChopAssoc SChopstar-1L SChopstar-imp-empty int-iffI inteq-reflection*)
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *SChopstarMore-induct-lvar-equiv*:
$(\vdash (schopstar\ f) \frown g \longrightarrow g) = (\vdash (f \wedge more) \frown g \longrightarrow g)$
using *SChopstar-inductL-var-equiv*[*of LIFT f* $\wedge$ *more g*]
SChopstar-and-more[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-eq*:
assumes $\vdash f;g = g$
shows $\vdash (wpowerstar\ f);g \longrightarrow g$
using *assms*
using *WPowerstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstar-induct-lvar-eq*:
assumes $\vdash (f) \frown g = g$
shows $\vdash (schopstar\ f) \frown g \longrightarrow g$
using *assms*
using *SChopstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstarMore-induct-lvar-eq*:
assumes $\vdash (f \wedge more) \frown g = g$
shows $\vdash (schopstar\ f) \frown g \longrightarrow g$
using *assms SChopstar-induct-lvar-eq*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-eq2*:
assumes $\vdash f;g = g$
shows $\vdash (wpowerstar\ f);g = g$
using *assms*
by (*meson ChopImpChop EmptyChop Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-eq lift-imp-trans*)

lemma *SChopstar-induct-lvar-eq2*:
assumes $\vdash (f) \frown g = g$
shows $\vdash (schopstar\ f) \frown g = g$
using *assms*
by (*metis AndSChopB EmptySChop Prop10 SChopstar-imp-empty SChopstar-induct-lvar int-eq int-iffD1 int-iffI*)

lemma *SChopstarMore-induct-lvar-eq2*:
assumes $\vdash (f \wedge more) \frown g = g$
shows $\vdash (schopstar\ f) \frown g = g$
using *assms SChopstar-induct-lvar-eq2*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-empty*:
assumes $\vdash$ *empty* $\vee$ *f* ; *g* $\longrightarrow$ *g*
shows $\vdash$ (*wpowerstar f*) $\longrightarrow$ *g*
using *assms*
by (*metis ChopEmpty WPowerstarInductL inteq-reflection*)

lemma *SChopstar-induct-lvar-empty*:
assumes $\vdash$ *empty* $\vee$ (*f*) $\frown$ *g* $\longrightarrow$ *g*
shows $\vdash$ (*schopstar f*) $\longrightarrow$ *g*
using *assms*
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb WPowerstar-induct-lvar-empty inteq-reflection*

schop-d-def schopstar-d-def)

lemma *SChopstarMore-induct-lvar-empty*:
assumes $\vdash$ *empty* $\vee$ (*f* $\wedge$ *more*) $\frown$ *g* $\longrightarrow$ *g*
shows $\vdash$ (*schopstar f*) $\longrightarrow$ *g*
using *assms SChopstar-induct-lvar-empty*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-star*:
assumes $\vdash$ *f* ; (*wpowerstar g*) $\longrightarrow$ (*wpowerstar g*)
shows $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*wpowerstar g*)
using *assms*
by (*meson Prop02 WPowerstar-imp-empty WPowerstar-induct-lvar-empty*)

lemma *SChopstar-induct-lvar-star*:
assumes $\vdash$ (*f*) $\frown$ (*schopstar g*) $\longrightarrow$ (*schopstar g*)
shows $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar g*)
using *assms*
by (*meson Prop02 SChopstar-imp-empty SChopstar-induct-lvar-empty*)

lemma *SChopstarMore-induct-lvar-star*:
assumes $\vdash$ (*f* $\wedge$ *more*) $\frown$ (*schopstar g*) $\longrightarrow$ (*schopstar g*)
shows $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar g*)
using *assms* **using** *SChopstar-induct-lvar-star*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-leq*:
assumes $\vdash$ (*h* $\vee$ *f*;*g*) = *g*
shows $\vdash$ (*wpowerstar f*);*h* $\longrightarrow$ *g*
using *assms*
using *WPowerstarInductL int-iffD1* **by** *blast*

lemma *SChopstar-induct-leq*:

assumes $\vdash (h \vee (f)^\frown g) = g$
shows $\vdash (schopstar\ f)^\frown h \longrightarrow g$
using *assms*
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-leq inteq-reflection lift-imp-trans schop-d-def*)

lemma *SChopstarMore-induct-leq*:
assumes $\vdash (h \vee (f \wedge more)^\frown g) = g$
shows $\vdash (schopstar\ f)^\frown h \longrightarrow g$
using *assms SChopstar-induct-leq*[*of h LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-subid*:
assumes $\vdash f \longrightarrow empty$
shows $\vdash (wpowerstar\ f) = empty$
using *assms*
by (*meson ChopEmpty Prop02 Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-empty lift-imp-trans*)

lemma *SChopstar-subid*:
assumes $\vdash f \longrightarrow empty$
shows $\vdash (schopstar\ f) = empty$
using *assms*
by (*metis EmptyImpFinite FPowerstar-WPowerstar FPowerstar-more-absorb Prop10 WPowerstar-subid int-eq*
lift-imp-trans schopstar-d-def)

lemma *WPowerstar-subdist*:
$\vdash (wpowerstar\ f) \longrightarrow (wpowerstar\ (f \vee g))$
proof −
have *1*: $\vdash f;(wpowerstar\ (f \vee g)) \longrightarrow (f \vee g);(wpowerstar\ (f \vee g))$
using *OrChopEqv* **by** *fastforce*
have *2*: $\vdash (f \vee g);(wpowerstar\ (f \vee g)) \longrightarrow (wpowerstar\ (f \vee g))$
by (*simp add*: *WPowerstar-1L*)
show *?thesis*
using *1 2 WPowerstar-induct-lvar-star lift-imp-trans* **by** *blast*
qed

lemma *SChopstar-subdist*:
$\vdash (schopstar\ f) \longrightarrow (schopstar\ (f \vee g))$
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb FiniteOr WPowerstar-subdist int-eq schop-star-d-def*)

lemma *WPowerstar-subdist-var*:
$\vdash (wpowerstar\ f) \vee (wpowerstar\ g) \longrightarrow (wpowerstar\ (f \vee g))$
by (*metis Prop02 WPowerstar-subdist WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var*:
$\vdash (schopstar\ f) \vee (schopstar\ g) \longrightarrow (schopstar\ (f \vee g))$
by (*metis Prop02 SChopstar-subdist SChopstar-swap inteq-reflection*)

lemma *WPowerstar-iso*:
assumes $\vdash f \longrightarrow g$
shows $\vdash (wpowerstar\ f) \longrightarrow (wpowerstar\ g)$
by (*metis AndChopB Prop05 Prop10 WPowerstar-induct-lvar-star WPowerstarEqv assms inteq-reflection*)

lemma *SChopstar-iso*:
assumes $\vdash f \longrightarrow g$
shows $\vdash (schopstar\ f) \longrightarrow (schopstar\ g)$
using *assms*
by (*metis AndSChopB Prop10 SChopstar-1L SChopstar-induct-lvar-star inteq-reflection lift-imp-trans*)

lemma *WPowerstar-invol*:
$\vdash (wpowerstar\ (wpowerstar\ f)) = (wpowerstar\ f)$
proof −
have *1*: $\vdash\ (wpowerstar\ f);(wpowerstar\ f) = (wpowerstar\ f)$
by (*simp add*: *WPowerstar-trans-eq*)
have *2*: $\vdash (wpowerstar\ (wpowerstar\ f)) \longrightarrow (wpowerstar\ f)$
using *1 WPowerstar-induct-lvar-star int-iffD1* **by** *blast*
have *3*: $\vdash (wpowerstar\ (wpowerstar\ f));(wpowerstar\ (wpowerstar\ f)) \longrightarrow (wpowerstar\ (wpowerstar\ f))$
by (*simp add*: *WPowerstar-trans*)
have *4*: $\vdash f;(wpowerstar\ (wpowerstar\ f)) \longrightarrow\ (wpowerstar\ (wpowerstar\ f))$
using *WPowerstar-1L WPowerstar-inductL-var-equiv* **by** *blast*
have *5*: $\vdash (wpowerstar\ f) \longrightarrow\ (wpowerstar\ (wpowerstar\ f))$
by (*simp add*: *4 WPowerstar-induct-lvar-star*)
show *?thesis*
by (*simp add*: *2 5 Prop11*)
qed

lemma *SChopstar-invol*:
$\vdash (schopstar\ (schopstar\ f)) = (schopstar\ f)$
by (*meson Prop11 SChopstar-1L SChopstar-inductL-var-equiv SChopstar-induct-lvar-star*)

lemma *WPowerstar-star2*:
$\vdash (wpowerstar\ (empty \vee f)) = (wpowerstar\ f)$
by (*meson EmptyOrChopEqv Prop02 Prop03 Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-empty*

WPowerstarEqv lift-imp-trans)

lemma *SChopstar-star2*:
$\vdash (schopstar\ (empty \vee f)) = (schopstar\ f)$
by (*metis EmptyImpFinite FiniteOr Prop10 SChopstar-WPowerstar WPowerstar-star2 int-eq*)

lemma *Chop-WPowerstar-Closure*:
assumes $\vdash f \longrightarrow (wpowerstar\ h)$
$\vdash g \longrightarrow (wpowerstar\ h)$
shows $\vdash f;g \longrightarrow (wpowerstar\ h)$
proof −
have *1*: $\vdash g \vee (wpowerstar\ h);(wpowerstar\ h) \longrightarrow (wpowerstar\ h)$

by (*metis Prop02 WPowerstar-1L WPowerstar-induct-lvar assms(2)*)
have *2*: $\vdash$ (*wpowerstar h*); *g* $\longrightarrow$ (*wpowerstar h*)
by (*meson Prop02 WPowerstarInductL WPowerstar-1L assms(2)*)
have *3*: $\vdash$ *f*;*g* $\longrightarrow$ (*wpowerstar h*);*g*
by (*simp add*: *LeftChopImpChop assms(1)*)
show *?thesis*
using *2 3 lift-imp-trans* **by** *blast*
qed

lemma *SChop-SChopstar-Closure*:
assumes $\vdash$ *f* $\longrightarrow$ (*schopstar h*)
$\vdash$ *g* $\longrightarrow$ (*schopstar h*)
shows $\vdash$ *f*$\frown$*g* $\longrightarrow$ (*schopstar h*)
using *assms*
by (*metis AndSChopA Prop10 SChopAndB SChopstarMore-1L SChopstarMore-induct-lvar inteq-reflection lift-and-com lift-imp-trans*)

lemma *WPowerstar-wpowerstar-closure*:
assumes $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*wpowerstar h*)
shows $\vdash$ (*wpowerstar* (*wpowerstar f*)) $\longrightarrow$ (*wpowerstar h*)
using *assms*
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-induct-lvar-star*)

lemma *SChopstar-SChopstar-closure*:
assumes $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar h*)
shows $\vdash$ (*schopstar* (*schopstar f*)) $\longrightarrow$ (*schopstar h*)
using *assms*
by (*metis SChopstar-invol inteq-reflection*)

lemma *WPowerstar-closed-unfold*:
assumes $\vdash$ (*wpowerstar f*) = *f*
shows $\vdash$ *f* = (*empty* $\vee$ *f*;*f*)
using *assms*
by (*metis WPowerstarEqv int-eq*)

lemma *SChopstar-closed-unfold*:
assumes $\vdash$ (*schopstar f*) = *f*
shows $\vdash$ *f* = (*empty* $\vee$ (*f*)$\frown$*f*)
using *assms*
by (*metis SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *SChopstarMore-closed-unfold*:
assumes $\vdash$ (*schopstar f*) = *f*
shows $\vdash$ *f* = (*empty* $\vee$ (*f* $\wedge$ *more*)$\frown$*f*)
using *assms*
by (*metis SChopstarEqv int-eq schop-d-def*)

lemma *WPowerstar-ext*:
$\vdash$ *f* $\longrightarrow$ (*wpowerstar f*)

proof −
have *1*: $\vdash f \longrightarrow f;(wpowerstar\ f)$
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty int-eq*)
show *?thesis* **by** (*meson 1 WPowerstar-1L lift-imp-trans*)
qed

lemma *SChopstar-ext*:
$\vdash f \wedge finite \longrightarrow (schopstar\ f)$
by (*metis SChopstar-WPowerstar WPowerstar-ext inteq-reflection*)

lemma *SChopstarMore-ext*:
$\vdash f \wedge more \wedge finite \longrightarrow (schopstar\ f)$
by (*metis AndMoreAndFiniteEqvAndFmore FPowerstar-WPowerstar SChopstar-ext SChopstar-WPowerstar*

fmore-d-def int-eq schopstar-d-def)

lemma *WPowerstar-1R*:
$\vdash (wpowerstar\ f)\ ;\ f \longrightarrow (wpowerstar\ f)$
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-ext*)

lemma *SChopstar-1R*:
$\vdash (schopstar\ f) \frown (f \wedge finite) \longrightarrow (schopstar\ f)$
by (*simp add*: *SChop-SChopstar-Closure SChopstar-ext*)

lemma *SChopstarMore-1R*:
$\vdash (schopstar\ f) \frown (f \wedge fmore) \longrightarrow (schopstar\ f)$
by (*simp add*: *SChop-SChopstar-Closure SChopstarMore-ext fmore-d-def*)

lemma *WPowerstar-unfoldR*:
$\vdash empty \vee (wpowerstar\ f)\ ;\ f \longrightarrow (wpowerstar\ f)$
by (*meson Prop02 WPowerstar-1R WPowerstar-imp-empty*)

lemma *SChopstar-unfoldR*:
$\vdash empty \vee (schopstar\ f) \frown (f \wedge finite) \longrightarrow (schopstar\ f)$
by (*meson Prop02 SChopstar-1R SChopstar-imp-empty*)

lemma *SChopstarMore-unfoldR*:
$\vdash empty \vee (schopstar\ f) \frown (f \wedge fmore) \longrightarrow (schopstar\ f)$
by (*meson Prop02 SChopstarMore-1R SChopstar-imp-empty*)

lemma *WPowerstar-sim1*:
assumes $\vdash f\ ;\ h \longrightarrow h\ ;\ g$
shows $\vdash (wpowerstar\ f)\ ;h \longrightarrow h\ ;(wpowerstar\ g)$
proof −
have *1*: $\vdash (f\ ;h);(wpowerstar\ g) \longrightarrow (h\ ;\ g);(wpowerstar\ g)$
by (*simp add*: *LeftChopImpChop assms*)
have *2*: $\vdash (h\ ;\ g);(wpowerstar\ g) \longrightarrow h\ ;(wpowerstar\ g)$
by (*metis ChopAssoc RightChopImpChop WPowerstar-1L inteq-reflection*)
have *3*: $\vdash (f\ ;\ h);(wpowerstar\ g) \longrightarrow\ h\ ;(wpowerstar\ g)$

using *1 2 lift-imp-trans* **by** *blast*
have *4*: $\vdash$ (*wpowerstar g*) =(*empty* $\vee$ *g*;(*wpowerstar g*))
by (*simp add*: *WPowerstarEqv*)
have *41*: $\vdash$ *h*;(*empty* $\vee$ *g*;(*wpowerstar g*)) = (*h*;*empty* $\vee$ *h*;(*g*;(*wpowerstar g*)))
by (*simp add*: *ChopOrEqv*)
have *42*: $\vdash$ *h*;(*g*;(*wpowerstar g*)) = (*h*;*g*);(*wpowerstar g*)
by (*simp add*: *ChopAssoc*)
have *5*: $\vdash$ *h* ;(*wpowerstar g*) = (*h* $\vee$ (*h*;*g*);(*wpowerstar g*))
by (*metis 4 41 42 ChopEmpty inteq-reflection*)
have *6*: $\vdash$ *h* $\longrightarrow$ *h* ;(*wpowerstar g*)
using *5* **by** *fastforce*
have *7*: $\vdash$ *h* $\vee$ (*f* ; *h*);(*wpowerstar g*) $\longrightarrow$ *h* ;(*wpowerstar g*)
using *3 6 Prop02* **by** *blast*
show *?thesis*
using *WPowerstarInductL*[*of LIFT h f LIFT h* ;(*wpowerstar g*)]
by (*metis 3 6 ChopAssoc Prop02 inteq-reflection*)
qed

lemma *SChopstar-sim1*:
assumes $\vdash$ *f* $\frown$ *h* $\longrightarrow$ *h* $\frown$ *g*
shows $\vdash$ (*schopstar f*) $\frown$(*h* $\wedge$ *finite*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
proof −
have *1*: $\vdash$ (*f* $\frown$*h*)$\frown$(*schopstar g*) $\longrightarrow$ (*h* $\frown$ *g*)$\frown$(*schopstar g*)
by (*simp add*: *LeftSChopImpSChop assms*)
have *2*: $\vdash$ (*h* $\frown$ *g*)$\frown$(*schopstar g*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
by (*metis RightSChopImpSChop SChopAssoc SChopstar-1L inteq-reflection*)
have *3*: $\vdash$ (*f* $\frown$ *h*)$\frown$(*schopstar g*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
using *1 2 lift-imp-trans* **by** *blast*
have *4*: $\vdash$ (*schopstar g*) =(*empty* $\vee$ (*g* $\wedge$ *more*)$\frown$(*schopstar g*))
by (*simp add*: *SChopstarEqv schop-d-def*)
have *5*: $\vdash$ *h* $\frown$(*schopstar g*) = ((*h* $\wedge$ *finite*) $\vee$ (*h*$\frown$(*g* $\wedge$ *more*))$\frown$(*schopstar g*))
by (*metis ChopEmpty SChopAssoc SChopOrEqv SChopstarEqv int-eq schop-d-def*)
have *6*: $\vdash$ *h* $\wedge$ *finite* $\longrightarrow$ *h* $\frown$(*schopstar g*)
using *5* **by** *fastforce*
have *7*: $\vdash$ (*h* $\wedge$ *finite*) $\vee$ (*f* $\frown$ *h*)$\frown$(*schopstar g*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
using *3 6 Prop02* **by** *blast*
show *?thesis* **using** *7*
by (*metis 3 6 Prop10 SChopAndB SChopAssoc SChopstar-induct-lvar inteq-reflection lift-imp-trans*)
qed

lemma *SChopstarMore-sim1*:
assumes $\vdash$ (*f* $\wedge$ *more*) $\frown$ *h* $\longrightarrow$ *h* $\frown$ (*g* $\wedge$ *more*)
shows $\vdash$ (*schopstar f*) $\frown$(*h* $\wedge$ *finite*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
using *assms SChopstar-sim1*[*of LIFT f* $\wedge$ *more h LIFT g* $\wedge$ *more*]
by (*metis SChopstar-and-more int-eq*)

lemma *WPowerstar-Quasicomm-varA*:
assumes $\vdash$ (*g*; *f* $\longrightarrow$ *f* ; (*wpowerstar* (*f* $\vee$ *g*)))
shows $\vdash$ ((*wpowerstar g*);*f* $\longrightarrow$ *f* ; (*wpowerstar* (*f* $\vee$ *g*)))

proof −
have *0*: ⊢ (*wpowerstar* (*wpowerstar* (*f* ∨ *g*))) = (*wpowerstar* (*f* ∨ *g*))
by (*meson WPowerstar-1L WPowerstar-inductL-var-equiv WPowerstar-induct-lvar-star int-iffI*)
have *2*: ⊢ *f* ; *wpowerstar wpowerstar* (*f* ∨ *g*) =
f ; (*wpowerstar* (*f* ∨ *g*))
by (*simp add*: *0 RightChopEqvChop*)
have *4*: ⊢ (*wpowerstar g* ; *f* ⟶
f ; *wpowerstar* (*f* ∨ *g*)) =
(*wpowerstar g* ; *f* ⟶
f ; *wpowerstar wpowerstar* (*f* ∨ *g*))
using *2* **by** *fastforce*
show *?thesis*
using *4 WPowerstar-sim1*[*of LIFT g LIFT f LIFT* (*wpowerstar* (*f* ∨ *g*))]
by (*metis 0 assms int-eq*)
qed

lemma *SChopstar-Quasicomm-varA*:
assumes ⊢ ((*g*)⌢ ((*f*) ∧ *finite*) ⟶ (*f*) ⌢ (*schopstar* (*f* ∨ *g*)))
shows ⊢ ((*schopstar g*)⌢((*f*) ∧ *finite*) ⟶ (*f*) ⌢ (*schopstar* (*f* ∨ *g*)))
proof −
have *0*: ⊢ (*schopstar* (*schopstar* (*f* ∨ *g*))) = (*schopstar* (*f* ∨ *g*))
by (*simp add*: *SChopstar-invol*)
have *2*: ⊢((*f*) ∧ *finite*) ⌢ *schopstar schopstar* (*f* ∨ *g*) =
(*f*) ⌢ (*schopstar* (*f* ∨ *g*))
by (*metis* (*no-types*, *lifting*) *0 AndSChopA Prop11 Prop12 inteq-reflection itl-def*(*9*) *lift-and-com*)
have *3*: ⊢ (*schopstar* (*g*) ⌢ (((*f*) ∧ *finite*) ∧ *finite*)) =
((*schopstar g*) ⌢ ((*f*) ∧ *finite*))
by (*metis Prop12 RightSChopImpSChop int-iffD2 int-iffI lift-and-com*)
have *4*: ⊢ (*schopstar* (*g*) ⌢ (((*f*) ∧ *finite*)) ⟶
(*f*) ⌢ *schopstar* (*f* ∨ *g*)) =
(*schopstar* (*g*) ⌢ (((*f*) ∧ *finite*) ∧ *finite*) ⟶
((*f*) ∧ *finite*) ⌢ *schopstar schopstar* (*f* ∨ *g*))

using *2 3* **by** *fastforce*
show *?thesis*
using *4 SChopstar-sim1*[*of LIFT g LIFT* (*f*) ∧ *finite LIFT* (*schopstar* (*f* ∨ *g*))]
by (*metis 0 2 assms int-eq*)
qed

lemma *SChopstarMore-or-and*:
⊢ *schopstar* (*f* ∧ *more* ∨ *g* ∧ *more*) = (*schopstar* ((*f* ∨ *g*) ∧ *more*))
proof −
have *1*: ⊢ (*f* ∧ *more* ∨ *g* ∧ *more*) = ((*f* ∨ *g*) ∧ *more*)
by *fastforce*
show *?thesis*
by (*metis 1 SChopstardef int-eq*)
qed

lemma *SChopstar-QuasicommMore-varA*:
assumes ⊢ ((*g* ∧ *more*)⌢ ((*f* ∧ *more*) ∧ *finite*) ⟶ (*f* ∧ *more*) ⌢ (*schopstar* (*f* ∨ *g*)))

shows $\vdash ((schopstar\ g)\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))$
using *assms SChopstar-Quasicomm-varA*[*of LIFT g* $\wedge$ *more LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*] *SChopstar-and-more*[*of LIFT* $(f \vee g)$]
AndMoreAndFiniteEqvAndFmore[*of f*]
by (*metis SChopstarMore-or-and inteq-reflection*)

lemma *WPowerstar-Quasicomm-varB*:
assumes $\vdash ((wpowerstar\ g);f \longrightarrow f\ ;\ (wpowerstar\ (f \vee g)))$
shows $\vdash (g;\ f \longrightarrow f\ ;\ (wpowerstar\ (f \vee g)))$
proof −
have *1*: $\vdash g;\ f \longrightarrow (wpowerstar\ g);f$
by (*simp add*: *LeftChopImpChop WPowerstar-ext*)
show *?thesis*
using *1 assms lift-imp-trans* **by** *blast*
qed

lemma *SChopstar-Quasicomm-varB*:
assumes $\vdash ((schopstar\ g)\frown((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g)))$
shows $\vdash ((g)\frown ((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g)))$
proof −
have *1*: $\vdash (g)\frown ((f) \wedge finite) \longrightarrow (schopstar\ g)\frown((f) \wedge finite)$
by (*metis LeftChopImpChop Prop12 SChopstar-ext int-iffD2 lift-and-com schop-d-def*)
show *?thesis*
using *1 assms lift-imp-trans* **by** *blast*
qed

lemma *SChopstar-QuasicommMore-varB*:
assumes $\vdash ((schopstar\ g)\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))$
shows $\vdash ((g \wedge more)\frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))$
using *assms SChopstar-Quasicomm-varB*[*of LIFT g* $\wedge$ *more LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*] *SChopstar-and-more*[*of LIFT* $(f \vee g)$]
AndMoreAndFiniteEqvAndFmore[*of f*]
by (*metis SChopstarMore-or-and inteq-reflection*)

lemma *WPowerstar-Quasicomm-var*:
$(\vdash (g;\ f \longrightarrow f\ ;\ (wpowerstar\ (f \vee g)))\ ) =$
$(\vdash\ ((wpowerstar\ g);f \longrightarrow f\ ;\ (wpowerstar\ (f \vee g))))$
using *WPowerstar-Quasicomm-varA WPowerstar-Quasicomm-varB* **by** *blast*

lemma *SChopstar-Quasicomm-var*:
$(\vdash ((g)\frown ((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g)))\ ) =$
$(\vdash\ ((schopstar\ g)\frown((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g))))$
using *SChopstar-Quasicomm-varA SChopstar-Quasicomm-varB* **by** *blast*

lemma *SChopstar-QuasicommMore-var*:
$(\vdash ((g \wedge more)\frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))\ ) =$
$(\vdash\ ((schopstar\ g)\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g))))$
using *SChopstar-QuasicommMore-varA SChopstar-QuasicommMore-varB* **by** *blast*

lemma *WPowerstar-slide1*:
$\vdash (wpowerstar\ (f\ ;\ g));f \longrightarrow f;(wpowerstar\ (g;f))$
by (*simp add*: *ChopAssoc WPowerstar-sim1 int-iffD2*)

lemma *SChopstar-slide1*:
$\vdash (schopstar\ (f \frown g))\frown(f \wedge finite) \longrightarrow f\frown(schopstar\ (g\frown f))$
using *SChopstar-sim1*
by (*metis SChopAssoc int-iffD2*)

lemma *WPowerstar-slide1-var1*:
$\vdash (wpowerstar\ f);f \longrightarrow f;(wpowerstar\ f)$
by (*meson Prop04 WPowerstar-sim1 int-iffD1 int-simps*(*27*))

lemma *SChopstar-slide1-var1*:
$\vdash (schopstar\ f)\frown(f \wedge finite) \longrightarrow f\frown(schopstar\ f)$
by (*simp add*: *SChopstar-sim1*)

lemma *SChopstarMore-slide1-var1*:
$\vdash (schopstar\ f)\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more)\frown(schopstar\ f)$
using *SChopstar-slide1-var1*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *wpowerstar-unfoldl-eq*:
$\vdash (empty \vee f;(wpowerstar\ f)) = (wpowerstar\ f)$
by (*meson Prop04 WPowerstar-1L WPowerstar-induct-lvar-star WPowerstarEqv int-iffI*)

lemma *SChopstar-unfoldl-eq*:
$\vdash (empty \vee f\frown(schopstar\ f)) = (schopstar\ f)$
by (*metis SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *SChopstarMore-unfoldl-eq*:
$\vdash (empty \vee (f \wedge more)\frown(schopstar\ f)) = (schopstar\ f)$
using *SChopstar-unfoldl-eq*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc1-eq*:
$\vdash (empty \vee f \vee (wpowerstar\ f);(wpowerstar\ f)) = (wpowerstar\ f)$
by (*meson Prop02 Prop05 Prop11 WPowerstar-ext WPowerstar-imp-empty WPowerstar-trans-eq*)

lemma *SChopstar-rtc1-eq*:
$\vdash (empty \vee (f \wedge finite) \vee (schopstar\ f)\frown(schopstar\ f)) = (schopstar\ f)$
by (*meson EmptySChop LeftSChopImpSChop Prop02 Prop04 Prop05 Prop08 Prop11 SChop-SChopstar-Closure*

SChopstar-ext SChopstar-imp-empty)

lemma *SChopstarMore-rtc1-eq*:

$\vdash$ (*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\vee$ (*schopstar f*)$\frown$(*schopstar f*)) = (*schopstar f*)
using *SChopstar-rtc1-eq*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc1*:
$\vdash$ (*empty* $\vee$ *f* $\vee$ (*wpowerstar f*);(*wpowerstar f*)) $\longrightarrow$ (*wpowerstar f*)
by (*meson WPowerstar-rtc1-eq int-iffD1*)

lemma *SChopstar-rtc1*:
$\vdash$ (*empty* $\vee$ (*f* $\wedge$ *finite*) $\vee$ (*schopstar f*)$\frown$(*schopstar f*)) $\longrightarrow$ (*schopstar f*)
by (*meson SChopstar-rtc1-eq int-iffD1*)

lemma *SChopstarMore-rtc1*:
$\vdash$ (*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\vee$ (*schopstar f*)$\frown$(*schopstar f*)) $\longrightarrow$ (*schopstar f*)
using *SChopstar-rtc1*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc2*:
($\vdash$ *empty* $\vee$ *f*;*f* $\longrightarrow$ *f*) = ($\vdash$ *f* = (*wpowerstar f*))
proof −
have *1*: ($\vdash$ *empty* $\vee$ *f*;*f* $\longrightarrow$ *f*) $\Longrightarrow$ ($\vdash$ *f* = (*wpowerstar f*))
using *WPowerstar-induct-lvar-empty*[*of f LIFT f*]
by (*simp add*: *WPowerstar-ext int-iffI*)
have *2*: ($\vdash$ *f* = (*wpowerstar f*)) $\Longrightarrow$ ($\vdash$ *empty* $\vee$ *f*;*f* $\longrightarrow$ *f*)
by (*metis WPowerstar-unfoldR inteq-reflection*)
show *?thesis*
by (*metis 1 2 inteq-reflection*)
qed

lemma *SChopstar-rtc2*:
($\vdash$ *empty* $\vee$ (*f*)$\frown$(*f* $\wedge$ *finite*) $\longrightarrow$ (*f* $\wedge$ *finite*)) = ($\vdash$ (*f* $\wedge$ *finite*) = (*schopstar f*))
proof −
have *1*: ($\vdash$ *empty* $\vee$ (*f*)$\frown$(*f* $\wedge$ *finite*) $\longrightarrow$ (*f* $\wedge$ *finite*)) $\Longrightarrow$ ($\vdash$ (*f* $\wedge$ *finite*) = (*schopstar f*))
using *SChopstar-induct-lvar-empty*[*of f LIFT f* $\wedge$ *finite*]
by (*simp add*: *Prop11 SChopstar-ext*)
have *2*: ($\vdash$ (*f* $\wedge$ *finite*) = (*schopstar f*)) $\Longrightarrow$ ($\vdash$ *empty* $\vee$ (*f*)$\frown$(*f* $\wedge$ *finite*) $\longrightarrow$ (*f* $\wedge$ *finite*))
by (*metis Prop02 SChopstar-1L SChopstar-imp-empty inteq-reflection*)
show *?thesis*
by (*metis 1 2 inteq-reflection*)
qed

lemma *SChopstarMore-rtc2*:
($\vdash$ *empty* $\vee$ (*f* $\wedge$ *more*)$\frown$((*f* $\wedge$ *more*) $\wedge$ *finite*) $\longrightarrow$ ((*f* $\wedge$ *more*) $\wedge$ *finite*)) =
($\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) = (*schopstar f*))
using *SChopstar-rtc2*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *SChopstarMore-rtc2-alt*:
($\vdash$ *empty* $\vee$ (*f* $\wedge$ *more*)$\frown$(*f* $\wedge$ *finite*) $\longrightarrow$ (*f* $\wedge$ *finite*)) = ($\vdash$ (*f* $\wedge$ *finite*) = (*schopstar f*))

proof −
have *1*: $(\vdash \mathit{empty} \vee (f \wedge \mathit{more})^\frown(f \wedge \mathit{finite}) \longrightarrow (f \wedge \mathit{finite})) \Longrightarrow \ (\vdash (f \wedge \mathit{finite}) = (\mathit{schopstar}\ f))$
using *SChopstarMore-induct-lvar-empty*[*of f LIFT f* $\wedge$ *finite*]
by (*simp add*: *SChopstar-ext int-iffI*)
have *2*: $(\vdash (f \wedge \mathit{finite}) = (\mathit{schopstar}\ f)) \Longrightarrow \ (\vdash \mathit{empty} \vee (f \wedge \mathit{more})^\frown(f \wedge \mathit{finite}) \longrightarrow (f \wedge \mathit{finite}))$
by (*metis SChopstarMore-unfoldl-eq int-eq int-iffD1*)
show *?thesis*
by (*metis 1 2 inteq-reflection*)
qed

lemma *WPowerstar-rtc3*:
$(\vdash (\mathit{empty} \vee f;f) = f) = (\vdash f = (\mathit{wpowerstar}\ f))$
by (*metis WPowerstar-rtc2 int-iffD1 inteq-reflection wpowerstar-unfoldl-eq*)

lemma *SChopstar-rtc3*:
$(\vdash (\mathit{empty} \vee (f)^\frown(f \wedge \mathit{finite})) = (f \wedge \mathit{finite})) = (\vdash (f \wedge \mathit{finite}) = (\mathit{schopstar}\ f))$
by (*metis SChopstar-rtc2 SChopstar-unfoldl-eq int-iffD1 inteq-reflection*)

lemma *SChopstarMore-rtc3*:
$(\vdash (\mathit{empty} \vee (f \wedge \mathit{more})^\frown((f \wedge \mathit{more}) \wedge \mathit{finite})) = ((f \wedge \mathit{more}) \wedge \mathit{finite})) =$
$(\vdash ((f \wedge \mathit{more}) \wedge \mathit{finite}) = (\mathit{schopstar}\ f))$
using *SChopstar-rtc3*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *SChopstarMore-rtc3-alt*:
$(\vdash (\mathit{empty} \vee (f \wedge \mathit{more})^\frown(f \wedge \mathit{finite})) = (f \wedge \mathit{finite})) = (\vdash (f \wedge \mathit{finite}) = (\mathit{schopstar}\ f))$
by (*metis SChopstarMore-induct-lvar-empty SChopstarMore-unfoldl-eq SChopstar-ext int-iffD1 int-iffI inteq-reflection*)

lemma *WPowerstar-rtc-least*:
assumes $\vdash \mathit{empty} \vee f \ \vee g;g \longrightarrow g$
shows $\vdash (\mathit{wpowerstar}\ f) \longrightarrow g$
proof −
have *1*: $\vdash f \ \longrightarrow g$
using *assms* **by** *fastforce*
have *2*: $\vdash g;g \longrightarrow g$
using *assms* **by** *fastforce*
have *3*: $\vdash f;g \longrightarrow g\ ;g$
by (*metis 1 LeftChopImpChop*)
have *4*: $\vdash f\ ;g \longrightarrow \ g$
using *2 3 lift-imp-trans* **by** *blast*
have *5*: $\vdash \mathit{empty} \longrightarrow g$
using *assms* **by** *fastforce*
show *?thesis*
by (*meson 4 5 Prop02 WPowerstar-induct-lvar-empty*)
qed

lemma *SChopstar-rtc-least*:

assumes $\vdash$ *empty* $\vee$ (*f* $\wedge$ *finite*) $\vee$ (*g*)$\frown$(*g*) $\longrightarrow$ (*g*)
shows $\vdash$ (*schopstar f*) $\longrightarrow$ *g*
proof −
have *1*: $\vdash$ (*f* $\wedge$ *finite*) $\longrightarrow$ (*g* $\wedge$ *finite*)
using *assms* **by** *fastforce*
have *2*: $\vdash$ (*g*)$\frown$(*g*) $\longrightarrow$ *g*
using *assms* **by** *fastforce*
have *3*: $\vdash$ (*f*)$\frown$(*g*) $\longrightarrow$ (*g*) $\frown$(*g*)
by (*metis 1 LeftChopImpChop schop-d-def*)
have *4*: $\vdash$ (*f*)$\frown$(*g*) $\longrightarrow$ *g*
using *2 3 lift-imp-trans* **by** *blast*
have *5*: $\vdash$ *empty* $\longrightarrow$ *g*
using *assms* **by** *fastforce*
show *?thesis*
by (*meson 4 5 Prop02 SChopstar-induct-lvar-empty*)
qed

lemma *SChopstarMore-rtc-least*:
assumes $\vdash$ *empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\vee$ (*g*)$\frown$(*g*) $\longrightarrow$ (*g*)
shows $\vdash$ (*schopstar f*) $\longrightarrow$ *g*
using *assms SChopstar-rtc-least*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc-least-eq*:
assumes $\vdash$ (*empty* $\vee$ *f* $\vee$ *g*;*g*) = *g*
shows $\vdash$ (*wpowerstar f*) $\longrightarrow$ *g*
using *assms*
using *WPowerstar-rtc-least int-iffD1* **by** *blast*

lemma *SChopstar-rtc-least-eq*:
assumes $\vdash$ (*empty* $\vee$ (*f* $\wedge$ *finite*) $\vee$ (*g*)$\frown$(*g*)) = (*g*)
shows $\vdash$ (*schopstar f*) $\longrightarrow$ *g*
using *SChopstar-rtc-least assms int-iffD1* **by** *blast*

lemma *SChopstarMore-rtc-least-eq*:
assumes $\vdash$ (*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\vee$ (*g*)$\frown$(*g*)) = (*g*)
shows $\vdash$ (*schopstar f*) $\longrightarrow$ *g*
using *assms SChopstar-rtc-least-eq*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-subdist-var-1*:
$\vdash$ *f* $\longrightarrow$ (*wpowerstar* (*f* $\vee$ *g*))
by (*meson WPowerstar-ext WPowerstar-subdist lift-imp-trans*)

lemma *SChopstar-subdist-var-1*:
$\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*meson SChopstar-ext SChopstar-subdist lift-imp-trans*)

lemma *WPowerstar-subdist-var-2*:

$\vdash f;g \longrightarrow (wpowerstar\ (f \vee g))$
by (*metis Chop-WPowerstar-Closure WPowerstar-subdist-var-1 WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var-2*:
$\vdash (f)\frown(g \wedge finite) \longrightarrow (schopstar\ (f \vee g))$
by (*metis Chop-WPowerstar-Closure SChopstar-swap SChopstar-WPowerstar SChopstar-subdist-var-1 inteq-reflection schop-d-def*)

lemma *SChopstarMore-subdist-var-2*:
$\vdash (f \wedge more)\frown((g \wedge more) \wedge finite) \longrightarrow (schopstar\ (f \vee g))$
using *SChopstar-subdist-var-2*[*of LIFT f* $\wedge$ *more LIFT g* $\wedge$ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* $(f \vee g)$]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-subdist-var-3*:
$\vdash (wpowerstar\ f);(wpowerstar\ g) \longrightarrow (wpowerstar\ (f \vee g))$
by (*metis Chop-WPowerstar-Closure WPowerstar-subdist WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var-3*:
$\vdash (schopstar\ f)\frown (schopstar\ g) \longrightarrow (schopstar\ (f \vee g))$
by (*metis SChop-SChopstar-Closure SChopstar-subdist SChopstar-swap inteq-reflection*)

lemma *WPowerstar-denest*:
$\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g)))$
proof −
have *1*: $\vdash f \longrightarrow (wpowerstar\ f)$
by (*simp add*: *WPowerstar-ext*)
have *2*: $\vdash (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty int-eq*)
have *3*: $\vdash f \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$
using *1 2* **by** *fastforce*
have *4*: $\vdash g \longrightarrow (wpowerstar\ g)$
by (*simp add*: *WPowerstar-ext*)
have *5*: $\vdash (wpowerstar\ g) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$
by (*meson EmptyChop LeftChopImpChop Prop11 WPowerstar-imp-empty lift-imp-trans*)
have *6*: $\vdash g \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$
using *4 5* **by** *fastforce*
have *7*: $\vdash f \vee g \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$
using *3 6* **by** *fastforce*
have *9*: $\vdash (wpowerstar\ (f \vee g)) \longrightarrow (wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g)))$
using *7 WPowerstar-iso* **by** *blast*
have *10*: $\vdash (wpowerstar\ f);(wpowerstar\ g) \longrightarrow (wpowerstar\ (f \vee g))$
by (*simp add*: *WPowerstar-subdist-var-3*)
have *11*: $\vdash (wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g))) \longrightarrow (wpowerstar\ (f \vee g))$
by (*simp add*: *10 Chop-WPowerstar-Closure WPowerstar-induct-lvar-star*)
show *?thesis* **using** *11 9 int-iffI* **by** *blast*
qed

lemma *SChopstar-denest*:
$\vdash$ (*schopstar* (*f* $\vee$ *g*)) = (*schopstar* ((*schopstar f*)$\frown$(*schopstar g*)))
proof −
have *1*: $\vdash$ (*f* $\wedge$ *finite*) $\longrightarrow$ (*schopstar f*)
by (*simp add*: *SChopstar-ext*)
have *2*: $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar f*)$\frown$(*schopstar g*)
by (*metis* (*no-types*, *lifting*) *AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop EmptyImpFinite FiniteAndEmptyEqvEmpty Prop02 Prop12 SChopAssoc SChopImpSChop SChopstar-1L SChopstar-imp-empty SChopstar-induct-lvar-empty inteq-reflection schop-d-def*)
have *3*: $\vdash$ (*f* $\wedge$ *finite*) $\longrightarrow$ (*schopstar f*)$\frown$(*schopstar g*)
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*g* $\wedge$ *finite*) $\longrightarrow$ (*schopstar g*)
by (*simp add*: *SChopstar-ext*)
have *5*: $\vdash$ (*schopstar g*) $\longrightarrow$ (*schopstar f*)$\frown$(*schopstar g*)
by (*meson EmptySChop LeftSChopImpSChop Prop11 SChopstar-imp-empty lift-imp-trans*)
have *6*: $\vdash$ (*g* $\wedge$ *finite*) $\longrightarrow$ (*schopstar f*)$\frown$(*schopstar g*)
using *4 5* **by** *fastforce*
have *7*: $\vdash$ (*f* $\wedge$ *finite*) $\vee$ (*g* $\wedge$ *finite*) $\longrightarrow$ (*schopstar f*)$\frown$(*schopstar g*)
using *3 6* **by** *fastforce*
have *8*: $\vdash$ ((*f* $\wedge$ *finite*) $\vee$ (*g* $\wedge$ *finite*)) = ((*f* $\vee$ *g*) $\wedge$ *finite*)
by *fastforce*
have *9*: $\vdash$ (*schopstar* (*f* $\vee$ *g*)) $\longrightarrow$ (*schopstar* ((*schopstar f*)$\frown$(*schopstar g*)))
by (*metis* (*no-types*, *opaque-lifting*) *7 8 ChopEmpty EmptySChop FPowerstar-WPowerstar FPowerstar-more-absorb SChopAssoc SChopstar-iso inteq-reflection schop-d-def schopstar-d-def*)
have *10*: $\vdash$ (*schopstar f*)$\frown$(*schopstar g*) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*simp add*: *SChopstar-subdist-var-3*)
have *11*: $\vdash$ (*schopstar* ((*schopstar f*)$\frown$(*schopstar g*))) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*simp add*: *10 SChop-SChopstar-Closure SChopstar-induct-lvar-star*)
show *?thesis* **using** *11 9 int-iffI* **by** *blast*
qed

lemma *WPowerstar-or-var*:
$\vdash$ (*wpowerstar* ((*wpowerstar f*) $\vee$ (*wpowerstar g*))) = (*wpowerstar* (*f* $\vee$ *g*))
using *WPowerstar-denest*[*of f g*]
WPowerstar-denest[*of LIFT* (*wpowerstar f*)]
WPowerstar-invol[*of f*] *WPowerstar-invol*[*of g*]
by (*metis int-eq*)

lemma *SChopstar-or-var*:
$\vdash$ (*schopstar* ((*schopstar f*) $\vee$ (*schopstar g*))) = (*schopstar* (*f* $\vee$ *g*))
by (*metis* (*no-types*, *opaque-lifting*) *FPowerstar-WPowerstar FPowerstar-more-absorb FiniteOr SChopstar-invol WPowerstar-denest inteq-reflection schopstar-d-def*)

lemma *WPowerstar-denest-var*:
$\vdash$ (*wpowerstar f*) ; (*wpowerstar* (*g*;(*wpowerstar f*))) = (*wpowerstar* (*f* $\vee$ *g*))
proof −
have *1*: $\vdash$ *empty* $\longrightarrow$ (*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*)))
by (*metis AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop FiniteAndEmptyEqvEmpty WPowerstar-imp-empty*

inteq-reflection)
have *2*: $\vdash (f;(wpowerstar\ f));(wpowerstar\ (g;(wpowerstar\ f))) \longrightarrow$
$(wpowerstar\ f);(wpowerstar\ (g;(wpowerstar\ f)))$
by (*simp add*: *LeftChopImpChop WPowerstar-1L*)
have *3*: $\vdash (g;(wpowerstar\ f));(wpowerstar\ (g;(wpowerstar\ f))) \longrightarrow$
$(wpowerstar\ f);(wpowerstar\ (g;(wpowerstar\ f)))$
by (*metis EmptyChop LeftChopImpChop WPowerstar-1L WPowerstar-imp-empty inteq-reflection lift-imp-trans*)

have *4*: $\vdash ((f;(wpowerstar\ f));(wpowerstar\ (g;(wpowerstar\ f))) \vee$
$(g;(wpowerstar\ f));(wpowerstar\ (g;(wpowerstar\ f)))) =$
$((f \vee g);(wpowerstar\ f));(wpowerstar\ (g;(wpowerstar\ f)))$
by (*metis OrChopEqv int-eq*)
have *5*: $\vdash empty \vee ((f \vee g);(wpowerstar\ f));(wpowerstar\ (g;(wpowerstar\ f))) \longrightarrow$
$(wpowerstar\ f);(wpowerstar\ (g;(wpowerstar\ f)))$
using *1 2 3 4* **by** (*metis Prop02 int-eq*)
have *6*: $\vdash (wpowerstar\ (f \vee g)) \longrightarrow (wpowerstar\ f);(wpowerstar\ (g;(wpowerstar\ f)))$
by (*metis 5 ChopAssoc WPowerstar-induct-lvar-empty int-eq*)
have *7*: $\vdash (wpowerstar\ (g\ ;(wpowerstar\ f))) \longrightarrow (wpowerstar\ ((wpowerstar\ g);(wpowerstar\ f)))$
by (*simp add*: *LeftChopImpChop WPowerstar-ext WPowerstar-iso*)
have *8*: $\vdash (wpowerstar\ (g\ ;(wpowerstar\ f))) \longrightarrow (wpowerstar\ (f \vee g))$
by (*metis 7 WPowerstar-denest WPowerstar-swap inteq-reflection*)
have *9*: $\vdash (wpowerstar\ f) \longrightarrow\ (wpowerstar\ (f \vee g))$
by (*simp add*: *WPowerstar-subdist*)
have *10*:$\vdash (wpowerstar\ f)\ ;\ (wpowerstar\ (g;(wpowerstar\ f))) \longrightarrow (wpowerstar\ (f \vee g))$
using *8 9 Chop-WPowerstar-Closure* **by** *blast*
show *?thesis*
by (*simp add*: *10 6 int-iffI*)
qed

lemma *SChopstar-denest-var*:
$\vdash (schopstar\ f) \frown (schopstar\ ((g)\frown(schopstar\ f))) = (schopstar\ (f \vee g))$
proof −
have *1*: $\vdash empty \longrightarrow (schopstar\ ((g)\frown(schopstar\ f)))$
by (*simp add*: *SChopstar-imp-empty*)
have *11*: $\vdash empty \longrightarrow (schopstar\ f)\frown(schopstar\ ((g)\frown(schopstar\ f)))$
by (*metis AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop EmptyImpFinite FiniteAndEmptyE-qvEmpty*
Prop12 SChopstar-imp-empty inteq-reflection itl-def(*9*))
have *2*: $\vdash ((f)\frown(schopstar\ f))\frown(schopstar\ ((g)\frown(schopstar\ f))) \longrightarrow$
$(schopstar\ f)\frown(schopstar\ ((g)\frown(schopstar\ f)))$
using *LeftSChopImpSChop SChopstar-1L* **by** *blast*
have *3*: $\vdash ((g)\frown(schopstar\ f))\frown(schopstar\ ((g\ )\frown(schopstar\ f))) \longrightarrow$
$(schopstar\ f)\frown(schopstar\ ((g)\frown(schopstar\ f)))$
by (*metis EmptySChop LeftSChopImpSChop SChopstar-1L SChopstar-imp-empty inteq-reflection lift-imp-trans*)
have *4*: $\vdash ((f \vee g)\frown(schopstar\ f))\frown(schopstar\ ((g)\frown(schopstar\ f))) =$
$(((f)\frown(schopstar\ f))\frown(schopstar\ ((g)\frown(schopstar\ f))) \vee$
$((g)\frown(schopstar\ f))\frown(schopstar\ ((g\ )\frown(schopstar\ f))))$

by (*metis OrSChopEqv inteq-reflection*)
have *5*: $\vdash empty \vee ((f \vee g)\frown(schopstar\ f))\frown(schopstar\ ((g)\frown(schopstar\ f))) \longrightarrow$

$(schopstar\ f)\frown(schopstar\ ((g)\frown(schopstar\ f)))$
using *11 2 3 4*
by (*metis Prop02 inteq-reflection*)
have *6*: $\vdash (schopstar\ (f \vee g)) \longrightarrow (schopstar\ f)\frown(schopstar\ ((g)\frown(schopstar\ f)))$
by (*metis 5 SChopAssoc SChopstar-induct-lvar-empty inteq-reflection*)
have *7*: $\vdash (schopstar\ (g \frown(schopstar\ f))) \longrightarrow (schopstar\ ((schopstar\ g)\frown(schopstar\ f)))$
by (*metis AndChopB ChopEmpty LeftChopImpChop Prop12 SChopstar-ext SChopstar-iso inteq-reflection schop-d-def*)
have *8*: $\vdash (schopstar\ (g \frown(schopstar\ f))) \longrightarrow (schopstar\ (f \vee g))$
by (*metis 7 SChopstar-denest SChopstar-swap inteq-reflection*)
have *9*: $\vdash (schopstar\ f) \longrightarrow (schopstar\ (f \vee g))$
by (*simp add*: *SChopstar-subdist*)
have *10*:$\vdash (schopstar\ f) \frown (schopstar\ ((g)\frown(schopstar\ f))) \longrightarrow (schopstar\ (f \vee g))$
by (*simp add*: *8 9 SChop-SChopstar-Closure*)
show *?thesis*
by (*simp add*: *10 6 int-iffI*)
qed

lemma *SChopstarMore-denest-var*:
$\vdash (schopstar\ f) \frown (schopstar\ ((g \wedge more)\frown(schopstar\ f))) = (schopstar\ (f \vee g))$
using *SChopstar-denest-var*[*of LIFT f* $\wedge$ *more LIFT g* $\wedge$ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* $(f \vee g)$]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-2*:
$\vdash (wpowerstar\ f);\ (wpowerstar\ (g;(wpowerstar\ f))) =$
$(wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g)))$
by (*metis WPowerstar-denest WPowerstar-denest-var int-eq*)

lemma *SChopstar-denest-var-2*:
$\vdash (schopstar\ f)\frown\ (schopstar\ (g\frown(schopstar\ f))) = (schopstar\ ((schopstar\ f)\frown(schopstar\ g)))$
by (*metis SChopstar-denest SChopstar-denest-var inteq-reflection*)

lemma *SChopstarMore-denest-var-2*:
$\vdash (schopstar\ f)\frown\ (schopstar\ ((g \wedge more)\frown(schopstar\ f))) = (schopstar\ ((schopstar\ f)\frown(schopstar\ g)))$
using *SChopstar-denest-var-2*[*of f LIFT g* $\wedge$ *more*] *SChopstar-and-more*[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-3*:
$\vdash (wpowerstar\ f);\ (wpowerstar\ ((wpowerstar\ g);(wpowerstar\ f))) =$
$(wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g)))$
by (*metis WPowerstar-denest WPowerstar-denest-var WPowerstar-ext WPowerstar-induct-lvar-star WPowerstar-trans int-iffI inteq-reflection*)

lemma *SChopstar-denest-var-3*:
$\vdash (schopstar\ f)\frown\ (schopstar\ ((schopstar\ g)\frown(schopstar\ f))) = (schopstar\ ((schopstar\ f)\frown(schopstar\ g)))$
by (*metis SChopstar-denest-var-2 SChopstar-invol int-eq*)

lemma *WPowerstar-denest-var-4*:
$\vdash$ (*wpowerstar* ((*wpowerstar g*);(*wpowerstar f*))) =
(*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
by (*metis WPowerstar-denest WPowerstar-swap inteq-reflection*)

lemma *SChopstar-denest-var-4*:
$\vdash$ (*schopstar* ((*schopstar g*)⌢(*schopstar f*))) = (*schopstar* ((*schopstar f*)⌢(*schopstar g*)))
by (*metis SChopstar-denest SChopstar-swap inteq-reflection*)

lemma *WPowerstar-denest-var-5*:
$\vdash$ (*wpowerstar f*); (*wpowerstar* (*g*;(*wpowerstar f*))) =
(*wpowerstar g*); (*wpowerstar* (*f*;(*wpowerstar g*)))
by (*metis WPowerstar-denest-var WPowerstar-swap int-eq*)

lemma *SChopstar-denest-var-5*:
$\vdash$ (*schopstar f*)⌢ (*schopstar* (*g*⌢(*schopstar f*))) = (*schopstar g*)⌢ (*schopstar* (*f*⌢(*schopstar g*)))
by (*metis SChopstar-denest-var SChopstar-swap inteq-reflection*)

lemma *SChopstarMore-denest-var-5*:
$\vdash$ (*schopstar f*)⌢ (*schopstar* ((*g* ∧ *more*)⌢(*schopstar f*))) = (*schopstar g*)⌢ (*schopstar* (*f*⌢(*schopstar g*)))

using *SChopstar-denest-var-5*[*of f LIFT g* ∧ *more*]
SChopstar-and-more[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-6*:
$\vdash$ ((*wpowerstar f*);(*wpowerstar g*));(*wpowerstar* (*f* ∨ *g*)) = (*wpowerstar* (*f* ∨ *g*))
by (*metis ChopAssoc WPowerstar-denest WPowerstar-denest-var-3 inteq-reflection*)

lemma *SChopstar-denest-var-6*:
$\vdash$ ((*schopstar f*)⌢(*schopstar g*))⌢(*schopstar* (*f* ∨ *g*)) = (*schopstar* (*f* ∨ *g*))
by (*metis SChopAssoc SChopstar-denest SChopstar-denest-var-3 inteq-reflection*)

lemma *WPowerstar-denest-var-7*:
$\vdash$ (*wpowerstar* (*f* ∨ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*)) = (*wpowerstar* (*f* ∨ *g*))
proof −
have *1*: $\vdash$ (*wpowerstar* (*f* ∨ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*)) ⟶
(*wpowerstar* (*f* ∨ *g*)) ; (*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
by (*simp add*: *RightChopImpChop WPowerstar-ext*)
have *2*: $\vdash$ (*wpowerstar* (*f* ∨ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*)) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-subdist-var-3*)
have *3*: $\vdash$ *empty* ⟶ (*wpowerstar* (*f* ∨ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*))
by (*metis ChopEmpty ChopImpChop WPowerstar-imp-empty inteq-reflection*)
have *4*: $\vdash$ (*f* ∨ *g*) ; ((*wpowerstar* (*f* ∨ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*))) ⟶
((*wpowerstar* (*f* ∨ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*)))

by (*metis ChopAssoc LeftChopImpChop WPowerstar-1L inteq-reflection*)
have *5*: $\vdash$ *empty* $\vee$ (*f* $\vee$ *g*) ; ((*wpowerstar* (*f* $\vee$ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*))) $\longrightarrow$
((*wpowerstar* (*f* $\vee$ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*)))
using *3 4 Prop02* **by** *blast*
have *6*: $\vdash$ (*wpowerstar* (*f* $\vee$ *g*)) $\longrightarrow$ ((*wpowerstar* (*f* $\vee$ *g*)) ; ((*wpowerstar f*);(*wpowerstar g*)))
using *5 WPowerstar-induct-lvar-empty* **by** *blast*
show *?thesis* **using** *2 6 int-iffI* **by** *blast*
qed

lemma *SChopstar-denest-var-7*:
$\vdash$ (*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*)) = (*schopstar* (*f* $\vee$ *g*))
proof −
have *1*: $\vdash$ (*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*)) $\longrightarrow$
(*schopstar* (*f* $\vee$ *g*)) $\frown$ (*schopstar* ((*schopstar f*)$\frown$(*schopstar g*)))
by (*meson RightSChopImpSChop SChopstar-denest SChopstar-subdist-var-3 int-iffD1 lift-imp-trans*)
have *2*: $\vdash$ (*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*)) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*simp add*: *SChop-SChopstar-Closure SChopstar-subdist-var-3*)
have *3*: $\vdash$ *empty* $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*))
by (*metis EmptySChop SChopImpSChop SChopstar-imp-empty inteq-reflection*)
have *4*: $\vdash$ (*f* $\vee$ *g*) $\frown$ ((*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*))) $\longrightarrow$
((*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*)))
by (*metis LeftSChopImpSChop SChopAssoc SChopstar-1L inteq-reflection*)
have *5*: $\vdash$ *empty* $\vee$ (*f* $\vee$ *g*) $\frown$ ((*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*))) $\longrightarrow$
((*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*)))
using *3 4 Prop02* **by** *blast*
have *6*: $\vdash$ (*schopstar* (*f* $\vee$ *g*)) $\longrightarrow$ ((*schopstar* (*f* $\vee$ *g*)) $\frown$ ((*schopstar f*)$\frown$(*schopstar g*)))
using *5 SChopstar-induct-lvar-empty* **by** *blast*
show *?thesis* **using** *2 6 int-iffI* **by** *blast*
qed

lemma *WPowerstar-denest-var-8*:
$\vdash$ ((*wpowerstar f*);(*wpowerstar g*));(*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*))) =
(*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
by (*metis ChopAssoc WPowerstar-denest-var-3 int-eq*)

lemma *SChopstar-denest-var-8*:
$\vdash$ ((*schopstar f*)$\frown$(*schopstar g*))$\frown$(*schopstar* ((*schopstar f*)$\frown$(*schopstar g*))) =
(*schopstar* ((*schopstar f*)$\frown$(*schopstar g*)))
by (*metis SChopstar-denest SChopstar-denest-var-6 inteq-reflection*)

lemma *WPowerstar-denest-var-9*:
$\vdash$ (*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)));((*wpowerstar f*);(*wpowerstar g*)) =
(*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
by (*metis WPowerstar-denest WPowerstar-denest-var-7 inteq-reflection*)

lemma *SChopstar-denest-var-9*:
$\vdash$ (*schopstar* ((*schopstar f*)$\frown$(*schopstar g*)))$\frown$((*schopstar f*)$\frown$(*schopstar g*)) =
(*schopstar* ((*schopstar f*)$\frown$(*schopstar g*)))
by (*metis SChopstar-denest SChopstar-denest-var-7 inteq-reflection*)

lemma *WPowerstar-confluence*:
$(\vdash g\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)) =$
$(\vdash (wpowerstar\ g)\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)\ )$
proof −
have *1*: $(\vdash g\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)) \Longrightarrow$
$(\vdash (wpowerstar\ g)\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)\ )$
by (*metis Prop02 WPowerstar-imp-empty WPowerstar-rtc2 WPowerstar-sim1 WPowerstar-trans inteq-reflection*)
have *2*: $(\vdash (wpowerstar\ g)\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)\ ) \Longrightarrow$
$(\vdash g\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g))$
by (*metis AndChopB Prop10 WPowerstar-ext inteq-reflection lift-imp-trans*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstar-finite*:
$\vdash schopstar\ f \longrightarrow finite$
by (*metis EmptyImpFinite FiniteChopEqvDiamond FiniteChopFiniteEqvFinite Prop02 SChopImpDiamond*

SChopstar-induct-lvar-empty inteq-reflection)

lemma *SChopstar-confluence*:
$(\vdash g \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)) =$
$(\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)\ )$
proof −
have *1*: $(\vdash g \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)) \Longrightarrow$
$(\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)\ )$
using *SChopstar-sim1*[*of g LIFT* (*schopstar f*) *LIFT* (*schopstar g*)]
by (*metis Prop10 SChopstar-finite SChopstar-invol inteq-reflection*)
have *2*: $(\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)\ ) \Longrightarrow$
$(\vdash g \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g))$
by (*metis AndChopB Prop10 SChopstar-ext SChopstar-finite int-eq lift-imp-trans schop-d-def*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstarMore-confluence*:
$(\vdash (g \wedge more) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)) =$
$(\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)\ )$
using *SChopstar-confluence*[*of LIFT g* $\wedge$ *more f*]
SChopstar-and-more[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-church-rosser*:
assumes $\vdash (wpowerstar\ g)\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
shows $\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
proof −
have *1*: $\vdash ((wpowerstar\ f)\ ;\ (wpowerstar\ g));((wpowerstar\ f)\ ;\ (wpowerstar\ g)) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ f));((wpowerstar\ g)\ ;\ (wpowerstar\ g))$
by (*metis* (*no-types*, *lifting*) *ChopAssoc ChopImpChop WPowerstar-trans WPowerstar-trans-eq assms*

int-eq)
have *2*: $\vdash ((wpowerstar\ f)\ ;\ (wpowerstar\ g));((wpowerstar\ f)\ ;\ (wpowerstar\ g)) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
by (*metis 1 WPowerstar-trans-eq inteq-reflection*)
have *3*: $\vdash empty \longrightarrow ((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
by (*metis 2 WPowerstar-denest-var-9 WPowerstar-imp-empty WPowerstar-induct-lvar int-eq lift-imp-trans*)
have *4*: $\vdash empty \vee ((wpowerstar\ f)\ ;\ (wpowerstar\ g));((wpowerstar\ f)\ ;\ (wpowerstar\ g)) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
using *2 3 Prop02* **by** *blast*
have *5*: $\vdash (wpowerstar\ ((wpowerstar\ f)\ ;\ (wpowerstar\ g))) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
using *4 WPowerstar-induct-lvar-empty* **by** *blast*
have *6*: $\vdash (wpowerstar\ (f \vee g)) \longrightarrow (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
by (*metis 5 WPowerstar-denest inteq-reflection*)
have *7*: $\vdash (wpowerstar\ f)\ ;\ (wpowerstar\ g) \longrightarrow (wpowerstar\ (f \vee g))$
by (*simp add*: *WPowerstar-subdist-var-3*)
show *?thesis*
by (*simp add*: *6 7 int-iffI*)
qed

lemma *SChopstar-church-rosser*:
assumes $\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
shows $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g)$
proof −
have *0*: $\vdash ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ f)) \frown ((schopstar\ g))$
by (*metis RightSChopImpSChop SChopAssoc assms inteq-reflection*)
have *01*: $\vdash (((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f))) \frown (schopstar\ g) \longrightarrow$
$(((schopstar\ f) \frown (schopstar\ f)) \frown ((schopstar\ g))) \frown (schopstar\ g)$
by (*simp add*: *0 LeftSChopImpSChop*)
have *1*: $\vdash ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f) \frown (schopstar\ g)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ f)) \frown ((schopstar\ g) \frown (schopstar\ g))$
by (*metis 01 SChopAssoc inteq-reflection*)
have *2*: $\vdash ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f) \frown (schopstar\ g)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ g))$
by (*metis* (*no-types*, *lifting*) *1 SChopstar-denest SChopstar-denest-var-3 int-eq int-simps*(*27*))
have *3*: $\vdash empty \longrightarrow ((schopstar\ f) \frown (schopstar\ g))$
by (*meson EmptySChop Prop11 SChopImpSChop SChopstar-imp-empty lift-imp-trans*)
have *4*: $\vdash empty \vee ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f) \frown (schopstar\ g)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ g))$
using *2 3 Prop02* **by** *blast*
have *5*: $\vdash (schopstar\ ((schopstar\ f) \frown (schopstar\ g))) \longrightarrow ((schopstar\ f) \frown (schopstar\ g))$
using *4 SChopstar-induct-lvar-empty* **by** *blast*
have *6*: $\vdash (schopstar\ (f \vee g)) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
by (*metis 5 SChopstar-denest inteq-reflection*)
have *7*: $\vdash (schopstar\ f) \frown (schopstar\ g) \longrightarrow (schopstar\ (f \vee g))$
by (*simp add*: *SChopstar-subdist-var-3*)
show *?thesis*
by (*simp add*: *6 7 int-iffI*)
qed

lemma *WPowerstar-church-rosser-var*:
assumes $\vdash g \;;\; (wpowerstar\ f) \longrightarrow (wpowerstar\ f) \;;\; (wpowerstar\ g)$
shows $\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ f) \;;\; (wpowerstar\ g)$
using *assms*
by (*simp add*: *WPowerstar-church-rosser WPowerstar-confluence*)

lemma *SChopstar-church-rosser-var*:
assumes $\vdash g \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
shows $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g)$
using *assms*
using *SChopstar-church-rosser SChopstar-confluence* **by** *blast*

lemma *SChopstarMore-church-rosser-var*:
assumes $\vdash (g \wedge more) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
shows $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g)$
using *assms SChopstar-church-rosser-var*[*of LIFT g* $\wedge$ *more LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of g*] *SChopstar-and-more*[*of f*]
SChopstar-and-more[*of LIFT* $(f \vee g)$]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-church-rosser-to-confluence*:
assumes $\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ f) \;;\; (wpowerstar\ g)$
shows $\vdash (wpowerstar\ g) \;;\; (wpowerstar\ f) \longrightarrow (wpowerstar\ f) \;;\; (wpowerstar\ g)$
using *assms*
by (*metis WPowerstar-subdist-var-3 WPowerstar-swap inteq-reflection*)

lemma *SChopstar-church-rosser-to-confluence*:
assumes $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g)$
shows $\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
using *assms*
by (*metis SChopstar-subdist-var-3 SChopstar-swap inteq-reflection*)

lemma *WPowerstar-church-rosser-equiv*:
$(\vdash (wpowerstar\ g) \;;\; (wpowerstar\ f) \longrightarrow (wpowerstar\ f) \;;\; (wpowerstar\ g)) =$
$(\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ f) \;;\; (wpowerstar\ g))$
using *WPowerstar-church-rosser WPowerstar-church-rosser-to-confluence* **by** *blast*

lemma *SChopstar-church-rosser-equiv*:
$(\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)) =$
$(\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g))$
using *SChopstar-church-rosser SChopstar-church-rosser-to-confluence* **by** *blast*

lemma *WPowerstar-confluence-to-local-confluence*:
assumes $\vdash \ (wpowerstar\ g) \;;\; (wpowerstar\ f) \longrightarrow (wpowerstar\ f) \;;\; (wpowerstar\ g)$
shows $\vdash g \;;\; f \longrightarrow (wpowerstar\ f) \;;\; (wpowerstar\ g)$

using *assms*
WPowerstar-church-rosser[*of g f*]
by (*metis WPowerstar-denest WPowerstar-denest-var-4 WPowerstar-subdist-var-2 inteq-reflection*)

lemma *SChopstar-confluence-to-local-confluence*:
assumes ⊢ (*schopstar g*) ⌢ (*schopstar f*) ⟶ (*schopstar f*) ⌢ (*schopstar g*)
shows ⊢ (*g*) ⌢ (*f* ∧ *finite*) ⟶ (*schopstar f*) ⌢ (*schopstar g*)
using *assms*
SChopstar-church-rosser[*of g f*]
by (*metis SChopstar-denest SChopstar-denest-var-4 SChopstar-subdist-var-2 inteq-reflection*)

lemma *SChopstarMore-confluence-to-local-confluence*:
assumes ⊢ (*schopstar g*) ⌢ (*schopstar f*) ⟶ (*schopstar f*) ⌢ (*schopstar g*)
shows ⊢ (*g* ∧ *more*) ⌢ (*f* ∧ *finite*) ⟶ (*schopstar f*) ⌢ (*schopstar g*)
using *assms*
SChopstar-confluence-to-local-confluence[*of LIFT g* ∧ *more f*]
SChopstar-and-more[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-sup-id-star1*:
assumes ⊢ *empty* ⟶ *f*
shows ⊢ *f*;(*wpowerstar f*) = (*wpowerstar f*)
using *assms*
by (*metis* (*no-types*, *lifting*) *AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop ChopOrEqv ChopOrImp*

FiniteAndEmptyEqvEmpty Prop02 WPowerstarEqv WPowerstar-1L WPowerstar-imp-empty int-iffI inteq-reflection)

lemma *SChopstar-sup-id-star1*:
assumes ⊢ *empty* ⟶ *f*
shows ⊢ *f*⌢(*schopstar f*) = (*schopstar f*)
using *assms*
by (*metis EmptyImpFinite Prop12 SChopstar-WPowerstar WPowerstar-sup-id-star1 inteq-reflection schop-d-def*)

lemma *WPowerstar-sup-id-star2*:
assumes ⊢ *empty* ⟶ *f*
shows ⊢ (*wpowerstar f*);*f* = (*wpowerstar f*)
using *assms*
by (*metis ChopEmpty ChopImpChop WPowerstar-1R int-eq int-iffD2 int-iffI*)

lemma *SChopstar-sup-id-star2*:
assumes ⊢ *empty* ⟶ *f*
shows ⊢ (*schopstar f*)⌢(*f* ∧ *finite*) = (*schopstar f*)
using *assms*
WPowerstar-sup-id-star2[*of LIFT* (*f* ∧ *finite*)] *SChopstar-WPowerstar*[*of f*]
by (*metis EmptyImpFinite Prop10 Prop12 SChopstar-finite inteq-reflection schop-d-def*)

lemma *WPowerstar-unfoldr-eq*:
⊢ (*empty* ∨ (*wpowerstar f*);*f*) = (*wpowerstar f*)
proof −

have *1*: $\vdash$ (*empty* $\vee$ (*wpowerstar f*);*f*) $\longrightarrow$ (*wpowerstar f*)
using *WPowerstar-unfoldR* **by** *auto*
have *2*: $\vdash$ (*empty* $\vee$ *f*;(*empty* $\vee$ (*wpowerstar f*);*f*)) =
(*empty* $\vee$ (*empty* $\vee$ *f*;(*wpowerstar f*));*f*)
by (*metis* (*no-types*, *lifting*) *ChopAssoc ChopEmpty ChopOrEqv EmptyOrChopEqv inteq-reflection*)
have *3*: $\vdash$ (*empty* $\vee$ (*empty* $\vee$ *f*;(*wpowerstar f*));*f*) =
(*empty* $\vee$ (*wpowerstar f*);*f*)
by (*metis 2 inteq-reflection wpowerstar-unfoldl-eq*)
have *4*: $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*empty* $\vee$ (*wpowerstar f*);*f*)
by (*metis 2 3 WPowerstar-induct-lvar-empty int-eq int-iffD1*)
show *?thesis*
using *1 4 int-iffI* **by** *blast*
qed

lemma *SChopstar-unfoldr-eq*:
$\vdash$ (*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*)) = (*schopstar f*)
proof −
have *1*: $\vdash$ (*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*)) $\longrightarrow$ (*schopstar f*)
using *SChopstar-unfoldR* **by** *auto*
have *2*: $\vdash$ (*empty* $\vee$ *f*$\frown$(*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*))) =
(*empty* $\vee$ (*empty* $\vee$ *f*$\frown$(*schopstar f*))$\frown$(*f* $\wedge$ *finite*))
by (*metis* (*no-types*, *lifting*) *ChopEmpty EmptyOrSChopEq SChopAssoc SChopOrEqv inteq-reflection itl-def*(*9*))
have *3*: $\vdash$ (*empty* $\vee$ (*empty* $\vee$ *f*$\frown$(*schopstar f*))$\frown$(*f* $\wedge$ *finite*)) =
(*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*))
by (*metis 2 SChopstar-unfoldl-eq inteq-reflection*)
have *4*: $\vdash$ (*schopstar f*) $\longrightarrow$ (*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*))
by (*metis 2 3 SChopstar-induct-lvar-empty int-eq int-iffD1*)
show *?thesis*
using *1 4 int-iffI* **by** *blast*
qed

lemma *SChopstarMore-unfoldr-eq*:
$\vdash$ (*empty* $\vee$ (*schopstar f*)$\frown$((*f* $\wedge$ *more*) $\wedge$ *finite*)) = (*schopstar f*)
using *SChopstar-unfoldr-eq*[*of LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-star-prod-unfold*:
$\vdash$ (*empty* $\vee$ *f*;((*wpowerstar* (*g*;*f*));*g*)) = (*wpowerstar* (*f* ; *g*))
proof −
have *1*: $\vdash$ (*wpowerstar* (*f* ; *g*)) = (*empty* $\vee$ (*wpowerstar* (*f*;*g*));(*f*;*g*))
by (*meson WPowerstar-unfoldR WPowerstar-unfoldr-eq int-iffD2 int-iffI*)
have *2*: $\vdash$ (*wpowerstar* (*f*;*g*));(*f*;*g*) $\longrightarrow$ *f*;((*wpowerstar* (*g*;*f*));*g*)
using *WPowerstar-slide1*[*of f g*]
by (*metis AndChopB ChopAssoc Prop10 int-eq*)
have *3*: $\vdash$ (*wpowerstar* (*f* ; *g*)) $\longrightarrow$ (*empty* $\vee$ *f*;((*wpowerstar* (*g*;*f*));*g*))
using *1 2* **by** *fastforce*
have *4*: $\vdash$ *f*;((*wpowerstar* (*g*;*f*));*g*) $\longrightarrow$ (*f*;*g*);(*wpowerstar* (*f* ; *g*))
using *WPowerstar-slide1*[*of g f*]

by (*metis ChopAssoc RightChopImpChop inteq-reflection*)
have *5*: $\vdash$ (*empty* $\vee$ *f*;((*wpowerstar* (*g*;*f*));*g*)) $\longrightarrow$
empty $\vee$ (*f*;*g*);(*wpowerstar* (*f* ; *g*))
using *4* **by** *fastforce*
have *6*: $\vdash$ (*empty* $\vee$ (*f*;*g*);(*wpowerstar* (*f* ; *g*))) $\longrightarrow$ (*wpowerstar* (*f* ; *g*))
by (*meson WPowerstarEqv int-iffD2*)
show *?thesis*
by (*meson 3 5 6 int-iffI lift-imp-trans*)
qed

lemma *FiniteImportSChopRight*:
$\vdash$ (*finite* $\wedge$ (*f* $\frown$ *g*)) = *f*$\frown$(*g* $\wedge$ *finite*)
by (*metis ChopEmpty SChopAssoc inteq-reflection lift-and-com schop-d-def*)

lemma *SChopstar-star-prod-unfold*:
$\vdash$ (*empty* $\vee$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*))) = (*schopstar* (*f* $\frown$ *g*))
proof −
have *1*: $\vdash$ (*schopstar* (*f* $\frown$ *g*)) = (*empty* $\vee$ (*schopstar* (*f*$\frown$*g*))$\frown$(*f*$\frown$(*g* $\wedge$ *finite*)))
by (*metis FiniteImportSChopRight SChopAndCommute SChopstar-unfoldr-eq inteq-reflection*)
have *2*: $\vdash$ (*schopstar* (*f*$\frown$*g*))$\frown$(*f*$\frown$(*g* $\wedge$ *finite*)) $\longrightarrow$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*))
using *SChopstar-slide1*[*of f g*]
by (*metis* (*no-types*, *opaque-lifting*) *ChopAndFiniteDist ChopEmpty LeftSChopImpSChop SChopAssoc inteq-reflection schop-d-def*)
have *3*: $\vdash$ (*schopstar* (*f* $\frown$ *g*)) $\longrightarrow$ (*empty* $\vee$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*)))
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*)) $\longrightarrow$ (*f*$\frown$*g*)$\frown$(*schopstar* (*f* $\frown$ *g*))
using *SChopstar-slide1*[*of g f*]
by (*metis RightSChopImpSChop SChopAssoc inteq-reflection*)
have *5*: $\vdash$ (*empty* $\vee$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*))) $\longrightarrow$
empty $\vee$ (*f*$\frown$*g*)$\frown$(*schopstar* (*f* $\frown$ *g*))
using *4* **by** *fastforce*
have *6*: $\vdash$ (*empty* $\vee$ (*f*$\frown$*g*)$\frown$(*schopstar* (*f* $\frown$ *g*))) $\longrightarrow$ (*schopstar* (*f* $\frown$ *g*))
by (*meson Prop02 SChopstar-1L SChopstar-imp-empty*)
show *?thesis*
by (*meson 3 5 6 int-iffI lift-imp-trans*)
qed

lemma *WPowerstar-slide* :
$\vdash$ (*wpowerstar* (*f*;*g*));*f* = *f*;(*wpowerstar* (*g*;*f*))
proof −
have *1*: $\vdash$ *f*;(*wpowerstar* (*g*;*f*)) = *f*;(*empty* $\vee$ *g*;((*wpowerstar* (*f*;*g*));*f*))
by (*metis RightChopEqvChop WPowerstar-star-prod-unfold inteq-reflection*)
have *2*: $\vdash$ *f*;(*empty* $\vee$ *g*;((*wpowerstar* (*f*;*g*));*f*)) =
(*f*;*empty* $\vee$ *f*;(*g*;((*wpowerstar* (*f*;*g*));*f*)))
by (*simp add*: *ChopOrEqv*)
have *3*: $\vdash$ *f*;*empty* = *empty*;*f*
by (*metis ChopEmpty EmptyChop int-eq*)
have *4*: $\vdash$ *f*;(*g*;((*wpowerstar* (*f*;*g*));*f*)) =
((*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*

by (*metis ChopAssoc int-eq*)
have *5*: ⊢ (*f*;*empty* ∨ *f*;(*g*;((*wpowerstar* (*f*;*g*));*f*))) =
(*empty*;*f* ∨ ((*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*)
using *3 4* **by** *fastforce*
have *6*: ⊢ (*empty*;*f* ∨ ((*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*) =
(*empty* ∨ ((*f*;*g*);(*wpowerstar* (*f*;*g*))));*f*
by (*meson OrChopEqv Prop11*)
have *7*: ⊢ *f*;(*empty* ∨ *g*;((*wpowerstar* (*f*;*g*));*f*)) =
(*empty* ∨ (*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*
by (*metis 2 3 4 6 inteq-reflection*)
have *8*: ⊢ (*empty* ∨ (*f*;*g*);(*wpowerstar* (*f*;*g*))) = (*wpowerstar* (*f*;*g*))
by (*simp add*: *wpowerstar-unfoldl-eq*)
show *?thesis* **by** (*metis 1 7 8 int-eq*)
qed

lemma *SChopstar-slide* :
⊢ (*schopstar* (*f*⌢*g*))⌢(*f* ∧ *finite*) = *f*⌢(*schopstar* (*g*⌢*f*))
proof −
have *1*: ⊢ *f*⌢(*schopstar* (*g*⌢*f*)) = *f*⌢(*empty* ∨ *g*⌢((*schopstar* (*f*⌢*g*))⌢(*f* ∧ *finite*)))
by (*metis RightSChopEqvSChop SChopstar-star-prod-unfold inteq-reflection*)
have *2*: ⊢ *f*⌢(*empty* ∨ *g*⌢((*schopstar* (*f*⌢*g*))⌢(*f* ∧ *finite*))) =
(*f*⌢*empty* ∨ *f*⌢(*g*⌢((*schopstar* (*f*⌢*g*))⌢(*f* ∧ *finite*))))
by (*simp add*: *SChopOrEqv*)
have *3*: ⊢ *f*⌢*empty* = *empty*⌢(*f* ∧ *finite*)
by (*metis DiamondEmptyEqvFinite EmptySChop FiniteAndEmptyEqvEmpty SChopAndCommute*
SChopAndEmptyEqvSChopAndEmpty TrueSChopEqvDiamond inteq-reflection)
have *4*: ⊢ *f*⌢(*g*⌢((*schopstar* (*f*⌢*g*))⌢(*f* ∧ *finite*))) =
((*f*⌢*g*)⌢(*schopstar* (*f*⌢*g*)))⌢(*f* ∧ *finite*)
by (*metis SChopAssoc inteq-reflection*)
have *5*: ⊢ (*f*⌢*empty* ∨ *f*⌢(*g*⌢((*schopstar* (*f*⌢*g*))⌢(*f* ∧ *finite*)))) =
(*empty*⌢(*f* ∧ *finite*) ∨ ((*f*⌢*g*)⌢(*schopstar* (*f*⌢*g*)))⌢(*f* ∧ *finite*))
using *3 4* **by** *fastforce*
have *6*: ⊢ (*empty*⌢(*f* ∧ *finite*) ∨ ((*f*⌢*g*)⌢(*schopstar* (*f*⌢*g*)))⌢(*f* ∧ *finite*)) =
(*empty* ∨ ((*f*⌢*g*)⌢(*schopstar* (*f*⌢*g*))))⌢(*f* ∧ *finite*)
by (*meson OrSChopEqv Prop11*)
have *7*: ⊢ *f*⌢(*empty* ∨ *g*⌢((*schopstar* (*f*⌢*g*))⌢(*f* ∧ *finite*))) =
(*empty* ∨ (*f*⌢*g*)⌢(*schopstar* (*f*⌢*g*)))⌢(*f* ∧ *finite*)
by (*metis 2 3 4 6 inteq-reflection*)
have *8*: ⊢ (*empty* ∨ (*f*⌢*g*)⌢(*schopstar* (*f*⌢*g*))) = (*schopstar* (*f*⌢*g*))
by (*simp add*: *SChopstar-unfoldl-eq*)
show *?thesis* **by** (*metis 1 7 8 int-eq*)
qed

lemma *WPowerstar-slide-var* :
⊢ (*wpowerstar f*) ; *f* = *f*;(*wpowerstar f*)
by (*metis EmptyOrChopEqv Prop11 WPowerstarInductR WPowerstar-slide1-var1 WPowerstar-unfoldr-eq*
int-eq)

lemma *SChopstar-slide-var* :
⊢ (*schopstar f*) ⌢ (*f* ∧ *finite*) = *f*⌢(*schopstar f*)

using *WPowerstar-slide-var*
by (*metis* (*no-types*, *lifting*) *FPowerstar-WPowerstar Prop10 SChopstar-finite SChopstar-WPowerstar int-eq schop-d-def*)

lemma *SChopstarMore-slide-var* :
$\vdash$ (*schopstar f*) $\frown$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) = (*f* $\wedge$ *more*)$\frown$(*schopstar f*)
using *SChopstar-slide-var*[*of LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-or-unfold-var*:
$\vdash$ (*empty* $\vee$ (*wpowerstar f*);((*wpowerstar* (*f* $\vee$ *g*));(*wpowerstar g*))) = (*wpowerstar* (*f* $\vee$ *g*))
by (*metis ChopAssoc WPowerstar-denest WPowerstar-denest-var-4 WPowerstar-slide inteq-reflection wpowerstar-unfoldl-eq*)

lemma *SChopstar-or-unfold-var*:
$\vdash$ (*empty* $\vee$ (*schopstar f*)$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*schopstar g*))) = (*schopstar* (*f* $\vee$ *g*))
proof −
have *1*: $\vdash$ (*schopstar f*)$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*schopstar g*)) =
(*schopstar f*)$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$((*schopstar g*) $\wedge$ *finite*))
by (*simp add*: *Prop10 RightSChopEqvSChop SChopstar-finite*)
have *2*: $\vdash$ (*schopstar f*)$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$((*schopstar g*) $\wedge$ *finite*)) =
(*schopstar* (*f* $\vee$ *g*))
by (*metis* (*no-types*, *lifting*) *SChopstar-denest-var SChopstar-denest-var-2 SChopstar-denest-var-3 SChopstar-slide SChopstar-swap int-eq*)
show *?thesis*
using *1 2 SChopstar-imp-empty* **by** *fastforce*
qed

lemma *WPowerstar-or-unfold*:
$\vdash$ ((*wpowerstar f*) $\vee$ (*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*)))) = (*wpowerstar* (*f* $\vee$ *g*))
proof −
have *1*: $\vdash$ (*wpowerstar* (*f* $\vee$ *g*)) = (*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*)))
by (*meson Prop11 WPowerstar-denest-var*)
have *2*: $\vdash$ (*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*))) =
(*wpowerstar f*);(*empty* $\vee$ (*g*;(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*))))
using *RightChopEqvChop WPowerstarEqv* **by** *blast*
have *3*: $\vdash$ (*wpowerstar f*);(*empty* $\vee$ (*g*;(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*)))) =
(*wpowerstar f*);(*empty* $\vee$ *g*;(*wpowerstar* (*f* $\vee$ *g*)))
by (*metis 1 2 ChopAssoc int-eq*)
have *4*: $\vdash$ (*wpowerstar f*);(*empty* $\vee$ *g*;(*wpowerstar* (*f* $\vee$ *g*))) =
((*wpowerstar f*);*empty* $\vee$ (*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*))))
using *ChopOrEqv* **by** *blast*
have *5*: $\vdash$ (*wpowerstar f*);*empty* = (*wpowerstar f*)
by (*simp add*: *ChopEmpty*)
have *6*: $\vdash$ ((*wpowerstar f*);*empty* $\vee$ (*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*)))) =
((*wpowerstar f*) $\vee$ (*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*))))
using *5* **by** *auto*
show *?thesis*

by (*metis 1 2 3 4 6 int-eq*)
qed

lemma *SChopstar-or-unfold*:
$\vdash ((schopstar\ f) \vee (schopstar\ f) \frown (g \frown (schopstar\ (f \vee g)))) = (schopstar\ (f \vee g))$
proof −
have *1*: $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ (g \frown (schopstar\ f)))$
by (*meson Prop11 SChopstar-denest-var*)
have *2*: $\vdash (schopstar\ f) \frown (schopstar\ (g \frown (schopstar\ f))) =$
$(schopstar\ f) \frown (empty \vee (g \frown (schopstar\ f)) \frown (schopstar\ (g \frown (schopstar\ f))))$
by (*meson Prop11 RightSChopEqvSChop SChopstar-unfoldl-eq*)
have *3*: $\vdash (schopstar\ f) \frown (empty \vee (g \frown (schopstar\ f)) \frown (schopstar\ (g \frown (schopstar\ f)))) =$
$(schopstar\ f) \frown (empty \vee g \frown (schopstar\ (f \vee g)))$
by (*metis 1 2 SChopAssoc inteq-reflection*)
have *4*: $\vdash (schopstar\ f) \frown (empty \vee g \frown (schopstar\ (f \vee g))) =$
$((schopstar\ f) \frown empty \vee (schopstar\ f) \frown (g \frown (schopstar\ (f \vee g))))$
using *SChopOrEqv* **by** *blast*
have *5*: $\vdash (schopstar\ f) \frown empty = (schopstar\ f)$
by (*metis ChopEmpty Prop10 SChopstar-finite inteq-reflection schop-d-def*)
have *6*: $\vdash ((schopstar\ f) \frown empty \vee (schopstar\ f) \frown (g \frown (schopstar\ (f \vee g)))) =$
$((schopstar\ f) \vee (schopstar\ f) \frown (g \frown (schopstar\ (f \vee g))))$
using *5* **by** *auto*
show *?thesis*
by (*metis 1 2 3 4 6 int-eq*)
qed

lemma *SChopstarMore-or-unfold*:
$\vdash ((schopstar\ f) \vee (schopstar\ f) \frown ((g \wedge more) \frown (schopstar\ (f \vee g)))) = (schopstar\ (f \vee g))$
using *SChopstar-or-unfold*[*of LIFT f* $\wedge$ *more LIFT g* $\wedge$ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* $(f \vee g)$]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-troeger*:
$\vdash (wpowerstar\ (f \vee g)); h =$
$(wpowerstar\ f); (g; ((wpowerstar\ (f \vee g)); h) \vee h)$
proof −
have *1*: $\vdash (wpowerstar\ (f \vee g)); h =$
$((wpowerstar\ f) \vee (wpowerstar\ f); (g; (wpowerstar\ (f \vee g)))); h$
using *WPowerstar-or-unfold*[*of f g*] **by** (*metis LeftChopEqvChop int-eq*)
have *2*: $\vdash ((wpowerstar\ f) \vee (wpowerstar\ f); (g; (wpowerstar\ (f \vee g)))); h =$
$((wpowerstar\ f); h \vee ((wpowerstar\ f); (g; (wpowerstar\ (f \vee g)))); h)$
by (*simp add*: *OrChopEqv*)
have *3*: $\vdash ((wpowerstar\ f); (g; (wpowerstar\ (f \vee g)))); h =$
$(wpowerstar\ f); (g; ((wpowerstar\ (f \vee g)); h))$
by (*metis ChopAssoc inteq-reflection*)
have *3*: $\vdash ((wpowerstar\ f); h \vee ((wpowerstar\ f); (g; (wpowerstar\ (f \vee g)))); h) =$
$(wpowerstar\ f); (h \vee g; ((wpowerstar\ (f \vee g)); h))$

by (*metis 3 ChopOrEqv inteq-reflection*)
have *4*: $\vdash (h \vee g;((wpowerstar\ (f \vee g));h)) =$
$(g;((wpowerstar\ (f \vee g));h) \vee h)$
by *fastforce*
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *SChopstar-troeger*:
$\vdash (schopstar\ (f \vee g))^\frown(h) =$
$(schopstar\ f)^\frown(g^\frown((schopstar\ (f \vee g))^\frown(h)) \vee (h))$
proof −
have *1*: $\vdash (schopstar\ (f \vee g))^\frown(h) =$
$((schopstar\ f) \vee (schopstar\ f)^\frown(g^\frown(schopstar\ (f \vee g))))^\frown(h)$
using *SChopstar-or-unfold*[*of f g*] **by** (*metis LeftSChopEqvSChop int-eq*)
have *2*: $\vdash ((schopstar\ f) \vee (schopstar\ f)^\frown(g^\frown(schopstar\ (f \vee g))))^\frown(h) =$
$((schopstar\ f)^\frown h \vee ((schopstar\ f)^\frown(g^\frown(schopstar\ (f \vee g))))^\frown h)$
by (*simp add*: *OrSChopEqv*)
have *3*: $\vdash ((schopstar\ f)^\frown(g^\frown(schopstar\ (f \vee g))))^\frown h =$
$(schopstar\ f)^\frown(g^\frown((schopstar\ (f \vee g))^\frown(h)))$
by (*metis SChopAssoc inteq-reflection*)
have *3*: $\vdash ((schopstar\ f)^\frown h \vee ((schopstar\ f)^\frown(g^\frown(schopstar\ (f \vee g))))^\frown h) =$
$(schopstar\ f)^\frown((h) \vee g^\frown((schopstar\ (f \vee g))^\frown(h)))$
by (*metis 3 SChopOrEqv inteq-reflection*)
have *4*: $\vdash ((h) \vee g^\frown((schopstar\ (f \vee g))^\frown(h))) =$
$(g^\frown((schopstar\ (f \vee g))^\frown(h)) \vee (h))$
by *fastforce*
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *SChopstarMore-troeger*:
$\vdash (schopstar\ (f \vee g))^\frown(h) =$
$(schopstar\ f)^\frown((g \wedge more)^\frown((schopstar\ (f \vee g))^\frown(h)) \vee (h))$
using *SChopstar-troeger*[*of LIFT f* $\wedge$ *more LIFT g* $\wedge$ *more h*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* $(f \vee g)$]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-square*:
$\vdash (wpowerstar\ (f;f)) \longrightarrow (wpowerstar\ f)$
proof −
have *1*: $\vdash (f;f);(wpowerstar\ f) \longrightarrow (wpowerstar\ f)$
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-ext*)
show *?thesis*
by (*simp add*: *1 WPowerstar-induct-lvar-star*)
qed

lemma *SChopstar-square*:

$\vdash$ (*schopstar* (*f*⌢*f*)) $\longrightarrow$ (*schopstar f*)
proof −
have *1*: $\vdash$ (*f*⌢*f*)⌢(*schopstar f*) $\longrightarrow$ (*schopstar f*)
by (*metis Prop05 RightSChopImpSChop SChopAssoc SChopstar-1L SChopstar-unfoldl-eq inteq-reflection*)
show *?thesis*
by (*simp add*: *1 SChopstar-induct-lvar-star*)
qed

lemma *WPowerstar-meyer-1*:
$\vdash$ (*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*)) = (*wpowerstar f*)
proof −
have *1*: $\vdash$ *f*;((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*))) =
f;(*empty*;(*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*)))
by (*simp add*: *OrChopEqv RightChopEqvChop*)
have *2*: $\vdash$ (*empty*;(*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*))) =
((*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*)))
by (*meson EmptyOrChopEqv OrChopEqv Prop04*)
have *3*: $\vdash$ *f*;(*empty*;(*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*))) =
f;((*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*)))
using *2 RightChopEqvChop* **by** *blast*
have *4*: $\vdash$ *f*;((*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*))) $\longrightarrow$
f;(*wpowerstar* (*f* ; *f*)) $\vee$ (*wpowerstar* (*f* ; *f*))
by (*metis ChopAssoc ChopOrImp Prop05 Prop08 int-eq int-iffD2 wpowerstar-unfoldl-eq*)
have *41*: $\vdash$ *f*;((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*))) $\longrightarrow$ *f*;(*wpowerstar* (*f*;*f*) $\vee$ *f*;*wpowerstar* (*f*;*f*))
using *EmptyOrChopEqv*[*of f LIFT* (*wpowerstar* (*f* ; *f*))]
by (*metis 1 Prop11 inteq-reflection*)
have *42*: $\vdash$ *f*;(*wpowerstar* (*f*;*f*) $\vee$ *f*;*wpowerstar* (*f*;*f*)) = (*f*;*wpowerstar* (*f*;*f*) $\vee$ *f*;(*f*;*wpowerstar* (*f*;*f*)))
using *ChopOrEqv*[*of f LIFT wpowerstar* (*f*;*f*) *LIFT f*;*wpowerstar* (*f*;*f*)]
by *auto*
have *43*: $\vdash$ (*f*;*wpowerstar* (*f*;*f*) $\vee$ *f*;(*f*;*wpowerstar* (*f*;*f*))) =
(*f*;*wpowerstar* (*f*;*f*) $\vee$ (*f*;*f*);*wpowerstar* (*f*;*f*))
using *ChopAssoc*[*of f f LIFT wpowerstar* (*f*;*f*)] **by** *auto*
have *44*: $\vdash$ (*f*;*wpowerstar* (*f*;*f*) $\vee$ (*f*;*f*);*wpowerstar* (*f*;*f*)) $\longrightarrow$ ((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*)))
using *wpowerstar-unfoldl-eq*[*of LIFT* (*f*;*f*)]
using *EmptyOrChopEqv* **by** *fastforce*
have *5*: $\vdash$ *f*;((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*))) $\longrightarrow$
((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*)))
by (*metis 1 3 42 43 44 int-eq*)
have *6*: $\vdash$ *empty* $\longrightarrow$ ((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*)))
using *EmptyOrChopEqv wpowerstar-unfoldl-eq* **by** *fastforce*
have *7*: $\vdash$ *empty* $\vee$ *f*;((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*))) $\longrightarrow$
((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*)))
using *5 6 Prop02* **by** *blast*
have *8*: $\vdash$ (*wpowerstar f*) $\longrightarrow$ ((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*)))
using *7 WPowerstar-induct-lvar-empty* **by** *blast*
have *9*: $\vdash$ ((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*))) $\longrightarrow$ (*empty* $\vee$ *f*) ;(*wpowerstar f*)
using *WPowerstar-square*[*of f*]
using *RightChopImpChop* **by** *blast*
have *10*: $\vdash$ (*empty* $\vee$ *f*) ;(*wpowerstar f*) $\longrightarrow$ (*wpowerstar f*)
by (*meson Prop02 WPowerstarInductR WPowerstar-1R WPowerstar-ext WPowerstar-imp-empty*)

show *?thesis*
by (*meson 10 8 9 int-iffI lift-imp-trans*)
qed

lemma *SChopstar-meyer-1*:
$\vdash$ (*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*)) = (*schopstar f*)
proof −
have *1*: $\vdash$ *f*$\frown$((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*))) =
f$\frown$(*empty*$\frown$(*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*)))
by (*simp add*: *OrSChopEqv RightSChopEqvSChop*)
have *2*: $\vdash$ (*empty*$\frown$(*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*))) =
((*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*)))
by (*meson EmptyOrSChopEqv OrSChopEqv Prop04*)
have *3*: $\vdash$ *f*$\frown$(*empty*$\frown$(*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*))) =
f$\frown$((*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*)))
using *2 RightSChopEqvSChop* **by** *blast*
have *4*: $\vdash$ *f*$\frown$((*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*))) $\longrightarrow$
(*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*))
using *SChopAssoc SChopOrEqv SChopstar-unfoldl-eq* **by** *fastforce*
have *41*: $\vdash$ ((*schopstar* (*f* $\frown$ *f*)) $\vee$ *f*$\frown$(*schopstar* (*f* $\frown$ *f*))) =
((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*)))
by (*metis 2 OrSChopEqv inteq-reflection*)
have *5*: $\vdash$ *f*$\frown$((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*))) $\longrightarrow$
((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*)))
by (*metis 4 41 int-eq*)
have *6*: $\vdash$ *empty* $\longrightarrow$ ((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*)))
using *EmptyOrSChopEqv SChopstar-imp-empty* **by** *fastforce*
have *7*: $\vdash$ *empty* $\vee$ *f*$\frown$((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*))) $\longrightarrow$
((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*)))
using *5 6 Prop02* **by** *blast*
have *8*: $\vdash$ (*schopstar f*) $\longrightarrow$ ((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*)))
using *7 SChopstar-induct-lvar-empty* **by** *blast*
have *9*: $\vdash$ ((*empty* $\vee$ *f*)$\frown$(*schopstar* (*f* $\frown$ *f*))) $\longrightarrow$ (*empty* $\vee$ *f*) $\frown$(*schopstar f*)
using *SChopstar-square*[*of f*]
using *RightSChopImpSChop* **by** *blast*
have *10*: $\vdash$ (*empty* $\vee$ *f*) $\frown$(*schopstar f*) $\longrightarrow$ (*schopstar f*)
by (*metis SChopstar-1L SChopstar-star2 inteq-reflection*)
show *?thesis*
by (*meson 10 8 9 int-iffI lift-imp-trans*)
qed

lemma *SChopstarMore-meyer-1*:
$\vdash$ (*empty* $\vee$ (*f* $\wedge$ *more*))$\frown$(*schopstar* (*f* $\frown$ *f*)) = (*schopstar f*)
proof −
have *1*: $\vdash$ (*f* $\wedge$ *more*)$\frown$((*empty* $\vee$ (*f* $\wedge$ *more*))$\frown$(*schopstar* (*f* $\frown$ *f*))) =
(*f* $\wedge$ *more*)$\frown$(*empty*$\frown$(*schopstar* (*f* $\frown$ *f*)) $\vee$ (*f* $\wedge$ *more*)$\frown$(*schopstar* (*f* $\frown$ *f*)))
by (*simp add*: *OrSChopEqv RightSChopEqvSChop*)
have *2*: $\vdash$ (*empty*$\frown$(*schopstar* (*f* $\frown$ *f*)) $\vee$ (*f* $\wedge$ *more*)$\frown$(*schopstar* (*f* $\frown$ *f*))) =
((*schopstar* (*f* $\frown$ *f*)) $\vee$ (*f* $\wedge$ *more*)$\frown$(*schopstar* (*f* $\frown$ *f*)))
by (*meson EmptyOrSChopEqv OrSChopEqv Prop04*)

have *3*: $\vdash (f \wedge more)^\frown(empty^\frown(schopstar\ (f^\frown f)) \vee (f \wedge more)^\frown(schopstar\ (f^\frown f))) =$
$(f \wedge more)^\frown((schopstar\ (f^\frown f)) \vee (f \wedge more)^\frown(schopstar\ (f^\frown f)))$
using *2 RightSChopEqvSChop* **by** *blast*
have *30*: $\vdash ((f \wedge more)^\frown(f \wedge more)) \longrightarrow f^\frown f$
by (*metis Prop11 Prop12 SChopImpSChop lift-and-com*)
have *31*: $\vdash ((f \wedge more)^\frown(f \wedge more))^\frown(schopstar\ (f^\frown f)) \longrightarrow (schopstar\ (f^\frown f))$
by (*metis 30 AndSChopA Prop05 Prop10 SChopstar-unfoldl-eq int-eq lift-and-com*)
have *4*: $\vdash (f \wedge more)^\frown((schopstar\ (f^\frown f)) \vee (f \wedge more)^\frown(schopstar\ (f^\frown f))) \longrightarrow$
$(f \wedge more)^\frown(schopstar\ (f^\frown f)) \vee (schopstar\ (f^\frown f))$
using *31 SChopAssoc SChopOrImp* **by** *fastforce*
have *5*: $\vdash (f \wedge more)^\frown((empty \vee (f \wedge more))^\frown(schopstar\ (f^\frown f))) \longrightarrow$
$((empty \vee (f \wedge more))^\frown(schopstar\ (f^\frown f)))$
by (*metis* (*no-types*, *opaque-lifting*) *4 EmptyOrSChopEqv Prop11 int-simps*(*33*)
inteq-reflection lift-and-com lift-imp-trans)
have *6*: $\vdash empty \longrightarrow ((empty \vee (f \wedge more))^\frown(schopstar\ (f^\frown f)))$
using *EmptyOrSChopEqv SChopstar-unfoldl-eq* **by** *fastforce*
have *7*: $\vdash empty \vee (f \wedge more)^\frown((empty \vee (f \wedge more))^\frown(schopstar\ (f^\frown f))) \longrightarrow$
$((empty \vee (f \wedge more))^\frown(schopstar\ (f^\frown f)))$
using *5 6 Prop02* **by** *blast*
have *8*: $\vdash (schopstar\ f) \longrightarrow ((empty \vee (f \wedge more))^\frown(schopstar\ (f^\frown f)))$
using *7 SChopstarMore-induct-lvar-empty* **by** *blast*
have *9*: $\vdash ((empty \vee (f \wedge more))^\frown(schopstar\ (f^\frown f))) \longrightarrow (empty \vee (f \wedge more))^\frown(schopstar\ f)$
using *SChopstar-square*[*of f*]
using *RightSChopImpSChop* **by** *blast*
have *10*: $\vdash (empty \vee (f \wedge more))^\frown(schopstar\ f) \longrightarrow (schopstar\ f)$
by (*metis SChopstar-1L SChopstar-and-more SChopstar-star2 inteq-reflection*)
show *?thesis*
by (*meson 10 8 9 int-iffI lift-imp-trans*)
qed

lemma *WPowerstar-tc*:
assumes $\vdash f;f \longrightarrow f$
shows $\vdash (wpowerstar\ f);f = f$
proof −
have *1*: $\vdash f \vee f;\ f \longrightarrow f$
using *assms* **by** *fastforce*
have *2*: $\vdash (wpowerstar\ f);f \longrightarrow f$
by (*simp add*: *WPowerstar-inductL-var-equiv assms*)
have *3*: $\vdash f \longrightarrow (wpowerstar\ f);f$
by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)
show *?thesis*
by (*simp add*: *2 3 Prop11*)
qed

lemma *SChopstar-tc*:
assumes $\vdash f^\frown f \longrightarrow f$
shows $\vdash (schopstar\ f)^\frown(f \wedge finite) = (f \wedge finite)$
proof −
have *1*: $\vdash f \vee f^\frown f \longrightarrow f$
using *assms* **by** *fastforce*

have *2*: $\vdash$ (*schopstar f*)⌢(*f* $\wedge$ *finite*) $\longrightarrow$ *f* $\wedge$ *finite*
by (*metis Prop12 SChopAndA SChopstar-1R SChopstar-finite SChopstar-induct-lvar assms lift-imp-trans*)
have *3*: $\vdash$ *f* $\wedge$ *finite*$\longrightarrow$ (*schopstar f*)⌢(*f* $\wedge$ *finite*)
by (*metis EmptySChop LeftSChopImpSChop SChopstar-imp-empty inteq-reflection*)
show *?thesis*
by (*simp add*: *2 3 Prop11*)
qed

lemma *WPowerstar-tc-eq*:
assumes $\vdash$ *f*;*f* = *f*
shows $\vdash$ (*wpowerstar f*);*f* = *f*
using *assms*
by (*simp add*: *WPowerstar-induct-lvar-eq2*)

lemma *SChopstar-tc-eq*:
assumes $\vdash$ *f*⌢*f* = *f*
shows $\vdash$ (*schopstar f*)⌢(*f* $\wedge$ *finite*) = (*f* $\wedge$ *finite*)
using *assms*
by (*simp add*: *SChopstar-tc int-iffD1*)

lemma *WPowerstar-boffa-var*:
assumes $\vdash$ *f*;*f* $\longrightarrow$ *f*
shows $\vdash$ (*wpowerstar f*) = (*empty* $\vee$ *f*)
proof −
have *1*: $\vdash$ (*wpowerstar f*) = (*empty* $\vee$ (*wpowerstar f*);*f*)
by (*metis WPowerstar-slide-var WPowerstarEqv int-eq*)
show *?thesis*
using *1 Prop06 WPowerstar-tc assms* **by** *blast*
qed

lemma *SChopstar-boffa-var*:
assumes $\vdash$ *f*⌢*f* $\longrightarrow$ *f*
shows $\vdash$ (*schopstar f*) = (*empty* $\vee$ (*f* $\wedge$ *finite*))
proof −
have *1*: $\vdash$ (*schopstar f*) = (*empty* $\vee$ (*schopstar f*)⌢(*f* $\wedge$ *finite*))
by (*meson SChopstar-unfoldR SChopstar-unfoldr-eq int-iffD2 int-iffI*)
show *?thesis*
using *1 Prop06 SChopstar-tc assms* **by** *blast*
qed

lemma *WPowerstar-boffa*:
assumes $\vdash$ *f*;*f* = *f*
shows $\vdash$ (*wpowerstar f*) = (*empty* $\vee$ *f*)
using *assms*
by (*simp add*: *WPowerstar-boffa-var int-iffD1*)

lemma *SChopstar-boffa*:
assumes $\vdash$ *f*⌢*f* = *f*
shows $\vdash$ (*schopstar f*) = (*empty* $\vee$ (*f* $\wedge$ *finite*))
using *assms*

by (*simp add*: *SChopstar-boffa-var int-iffD1*)

lemma *WPowerstar-sim2*:
assumes $\vdash h \ ; f \longrightarrow g \ ; h$
shows $\vdash h \ ; (wpowerstar\ f) \longrightarrow (wpowerstar\ g) \ ; h$
proof −
have *1*: $\vdash (wpowerstar\ g) \ ; (h \ ; f) \longrightarrow \ (wpowerstar\ g) \ ;(\ g \ ; h \ )$
by (*simp add*: *RightChopImpChop assms*)
have *2*: $\vdash (wpowerstar\ g) \ ;(\ g \ ; h \ ) \longrightarrow (wpowerstar\ g) \ ; h$
by (*metis ChopAssoc LeftChopImpChop WPowerstar-1L WPowerstar-slide-var inteq-reflection*)
have *3*: $\vdash (wpowerstar\ g) \ ; (h \ ; f) \longrightarrow (wpowerstar\ g) \ ; h$
using *1 2 lift-imp-trans* **by** *blast*
have *4*: $\vdash h \longrightarrow (wpowerstar\ g) \ ; h$
by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)
have *5*: $\vdash h \vee (wpowerstar\ g) \ ; (h \ ; f) \longrightarrow (wpowerstar\ g) \ ; h$
using *3 4 Prop02* **by** *blast*
show *?thesis*
by (*metis 5 ChopAssoc WPowerstarInductR inteq-reflection*)
qed

lemma *SChopstarInductR*:
assumes $\vdash g \vee h \frown f \longrightarrow h$
shows $\vdash g \frown schopstar\ f \longrightarrow h$
proof −
have *1*: $\vdash \quad g \wedge finite \ \longrightarrow h$
using *assms* **by** *fastforce*
have *2*: $\vdash h \frown f \longrightarrow h$
using *assms* **by** *fastforce*
have *3*: $\vdash g \wedge finite \vee \ h \frown f \longrightarrow h$
using *1 2* **by** *fastforce*
have *4*: $\vdash \ (g \wedge finite);fpowerstar\ f \longrightarrow h$
by (*metis 1 2 ChopSChopdef FPowerstarInductR OrFiniteInf Prop02 Prop03 inteq-reflection*)
show *?thesis*
using *FPowerstar-more-absorb*[*of f*] *4*
by (*metis int-eq schop-d-def schopstar-d-def*)
qed

lemma *SChopstar-sim2*:
assumes $\vdash h \frown f \longrightarrow g \frown h$
shows $\vdash h \frown (schopstar\ f) \longrightarrow (schopstar\ g) \frown h$
proof −
have *1*: $\vdash (schopstar\ g) \frown (h \frown (f)) \longrightarrow \ (schopstar\ g) \frown(\ g \frown h \ )$
by (*simp add*: *RightSChopImpSChop assms*)
have *2*: $\vdash (schopstar\ g) \frown(\ g \frown h \ ) \longrightarrow (schopstar\ g) \frown h$
by (*metis* (*no-types*, *lifting*) *ChopAssoc LeftChopImpChop Prop10 SChopstar-1L SChopstar-finite SChopstar-WPowerstar WPowerstar-slide-var inteq-reflection schop-d-def*)
have *3*: $\vdash (schopstar\ g) \frown (h \frown (f)) \longrightarrow (schopstar\ g) \frown h$
using *1 2 lift-imp-trans* **by** *blast*
have *4*: $\vdash h \longrightarrow (schopstar\ g) \frown h$

by (*meson EmptySChop LeftSChopImpSChop Prop11 SChopstar-imp-empty lift-imp-trans*)
have *5*: $\vdash h \vee (schopstar\ g) \frown (h \frown (f)) \longrightarrow (schopstar\ g) \frown h$
using *3 4 Prop02* **by** *blast*
show *?thesis*
by (*metis 5 SChopAssoc SChopstarInductR inteq-reflection*)
qed

lemma *SChopImpFinite*:
$\vdash f \longrightarrow finite \Longrightarrow \vdash g \frown f \longrightarrow finite$
by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop10 SChopSFinExportA SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection*)

lemma *SChopstar-sim2-finite*:
assumes $\vdash h \frown f \longrightarrow g \frown h$
shows $\vdash h \frown (schopstar\ f) \longrightarrow (schopstar\ g) \frown (h \wedge finite)$
using *assms SChopstar-sim2*
by (*metis FiniteImportSChopRight Prop12 SChopImpFinite SChopstar-finite inteq-reflection*)

lemma *WPowerstar-inductr-var*:
assumes $\vdash g\ ;\ f \longrightarrow g$
shows $\vdash g\ ;\ (wpowerstar\ f) \longrightarrow g$
using *assms*
by (*simp add*: *WPowerstarInductR*)

lemma *SChopstar-inductr-var*:
assumes $\vdash g \frown f \longrightarrow g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*simp add*: *SChopstarInductR*)

lemma *SChopstarMore-inductr-var*:
assumes $\vdash g \frown (f \wedge more) \longrightarrow g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-var inteq-reflection*)

lemma *SChopstar-finite-absorb*:
$\vdash schopstar\ (f \wedge finite) = schopstar\ f$
by (*metis Prop10 SChopstar-ext SChopstar-finite SChopstar-WPowerstar int-eq lift-imp-trans*)

lemma *SChopstar-inductr-finite-var*:
assumes $\vdash g \frown (f \wedge finite) \longrightarrow g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*metis SChopstar-finite-absorb SChopstar-inductr-var inteq-reflection*)

lemma *SChopstarMore-inductr-finite-var*:
assumes $\vdash g \frown ((f \wedge more) \wedge finite) \longrightarrow g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*

by (*metis SChopstar-and-more SChopstar-inductr-finite-var inteq-reflection*)

lemma *WPowerstar-inductr-var-equiv*:
$(\vdash g\ ;\ f \longrightarrow g\ ) = (\vdash g\ ;\ (wpowerstar\ f) \longrightarrow g)$
proof −
have *1*: $(\vdash g\ ;\ f \longrightarrow g\ ) \Longrightarrow (\vdash g\ ;\ (wpowerstar\ f) \longrightarrow g)$
by (*simp add*: *WPowerstar-inductr-var*)
have *2*: $(\vdash g\ ;\ (wpowerstar\ f) \longrightarrow g) \Longrightarrow (\vdash g\ ;\ f \longrightarrow g\ )$
by (*metis ChopAndB Prop10 WPowerstar-ext int-eq lift-imp-trans*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstar-inductr-var-equiv*:
$(\vdash g \frown (f \wedge finite) \longrightarrow g\ ) = (\vdash g \frown (schopstar\ f) \longrightarrow g)$
proof −
have *1*: $(\vdash g \frown (f \wedge finite) \longrightarrow g\ ) \Longrightarrow (\vdash g \frown (schopstar\ f) \longrightarrow g)$
by (*simp add*: *SChopstar-inductr-finite-var*)
have *2*: $(\vdash g \frown (schopstar\ f) \longrightarrow g) \Longrightarrow (\vdash g \frown (f \wedge finite) \longrightarrow g\ )$
by (*metis Prop10 Prop12 SChopAndB SChopstar-ext inteq-reflection*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstarMore-inductr-var-equiv*:
$(\vdash g \frown ((f \wedge more) \wedge finite) \longrightarrow g\ ) = (\vdash g \frown (schopstar\ f) \longrightarrow g)$
using *SChopstar-inductr-var-equiv*[*of g LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-sim3*:
assumes $\vdash h\ ;\ f = g\ ;\ h$
shows $\vdash h\ ;\ (wpowerstar\ f) = (wpowerstar\ g)\ ;h$
using *assms*
by (*simp add*: *Prop11 WPowerstar-sim1 WPowerstar-sim2*)

lemma *SChopstar-sim3*:
assumes $\vdash h \frown f = g \frown h$
shows $\vdash h \frown (schopstar\ f) = (schopstar\ g) \frown (h \wedge finite)$
using *SChopstar-sim1 SChopstar-sim2-finite*
by (*metis Prop11 assms*)

lemma *SChopstarMore-sim3*:
assumes $\vdash h \frown (f \wedge more) = (g \wedge more) \frown h$
shows $\vdash h \frown (schopstar\ f) = (schopstar\ g) \frown (h \wedge finite)$
by (*metis SChopstar-and-more SChopstar-sim3 assms inteq-reflection*)

lemma *WPowerstar-sim4*:
assumes $\vdash f\ ;\ g \longrightarrow g\ ;\ f$
shows $\vdash (wpowerstar\ f);(wpowerstar\ g) \longrightarrow (wpowerstar\ g)\ ;\ (wpowerstar\ f)$

using *assms*
by (*simp add*: *WPowerstar-sim1 WPowerstar-sim2*)

lemma *SChopstar-sim4*:
assumes $\vdash f \frown g \longrightarrow g \frown f$
shows $\vdash (schopstar\ f) \frown (schopstar\ g) \longrightarrow (schopstar\ g) \frown (schopstar\ f)$
using *assms*
by (*metis Prop10 SChopstar-finite SChopstar-sim1 SChopstar-sim2 inteq-reflection*)

lemma *WPowerstar-inductr-eq*:
assumes $\vdash (h \vee g\ ;\ f) = g$
shows $\vdash h;(wpowerstar\ f) \longrightarrow g$
using *assms*
using *WPowerstarInductR int-iffD1* **by** *blast*

lemma *SChopstar-inductr-eq*:
assumes $\vdash (h \vee g \frown f) = g$
shows $\vdash h \frown (schopstar\ f) \longrightarrow g$
using *assms* **using** *SChopstarInductR int-iffD1* **by** *blast*

lemma *SChopstarMore-inductr-eq*:
assumes $\vdash (h \vee g \frown (f \wedge more)) = g$
shows $\vdash h \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-eq inteq-reflection*)

lemma *WPowerstar-inductr-var-eq*:
assumes $\vdash (\ g\ ;\ f) = g$
shows $\vdash g;(wpowerstar\ f) \longrightarrow g$
using *assms*
using *WPowerstar-inductr-var int-iffD1* **by** *blast*

lemma *SChopstar-inductr-var-eq*:
assumes $\vdash (\ g \frown f) = g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
using *SChopstar-inductr-var int-iffD1* **by** *blast*

lemma *SChopstarMore-inductr-var-eq*:
assumes $\vdash (\ g \frown (f \wedge more)) = g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*simp add*: *SChopstarMore-inductr-var int-iffD1*)

lemma *WPowerstar-inductr-var-eq2*:
assumes $\vdash (\ g\ ;\ f) = g$
shows $\vdash g;(wpowerstar\ f) = g$
using *assms*
by (*metis Prop11 RightChopImpChop WPowerstar-ext WPowerstar-inductr-var-eq inteq-reflection*)

lemma *SChopstar-inductr-var-eq2*:
assumes $\vdash$ ($g \frown (f \wedge \mathit{finite})) = g$
shows $\vdash g \frown (\mathit{schopstar}\ f) = g$
using *assms*
by (*metis Prop11 RightSChopImpSChop SChopstar-ext SChopstar-inductr-finite-var inteq-reflection*)

lemma *SChopstarMore-inductr-var-eq2*:
assumes $\vdash$ ($g \frown ((f \wedge \mathit{more}) \wedge \mathit{finite})) = g$
shows $\vdash g \frown (\mathit{schopstar}\ f) = g$
using *assms*
SChopstar-inductr-var-eq2[*of g LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-bubble-sort*:
assumes $\vdash g\ ;\ f \longrightarrow f\ ;\ g$
shows $\vdash (\mathit{wpowerstar}\ (f \vee g)) = (\mathit{wpowerstar}\ f)\ ;\ (\mathit{wpowerstar}\ g)$
using *assms*
using *WPowerstar-church-rosser WPowerstar-sim4* **by** *blast*

lemma *SChopstar-bubble-sort*:
assumes $\vdash g \frown f \longrightarrow f \frown g$
shows $\vdash (\mathit{schopstar}\ (f \vee g)) = (\mathit{schopstar}\ f) \frown (\mathit{schopstar}\ g)$
using *assms* **by** (*simp add*: *SChopstar-church-rosser SChopstar-sim4*)

lemma *WPowerstar-indepence1*:
assumes $\vdash f\ ;\ g = \#\mathit{False}$
shows $\vdash (\mathit{wpowerstar}\ f);g = g$
using *assms*
by (*metis EmptyOrChopEqv WPowerstar-induct-lvar-eq2 WPowerstar-star2 assms int-eq int-simps*(*25*))

lemma *SChopRightFalse*:
$\vdash f \frown \#\mathit{False} = \#\mathit{False}$
by (*simp add*: *intI itl-defs*)

lemma *SChopstar-indepence1*:
assumes $\vdash f \frown g = \#\mathit{False}$
shows $\vdash (\mathit{schopstar}\ f) \frown g = g$
proof −
have *0* : $\vdash (\mathit{schopstar}\ f) = (\mathit{empty} \vee (\ \mathit{schopstar}\ f) \frown (f \wedge \mathit{finite}))$
by (*meson Prop11 SChopstar-unfoldr-eq*)
have *01*: $\vdash (\ \mathit{schopstar}\ f) \frown (f \wedge \mathit{finite}) = ((\mathit{schopstar}\ f) \frown f \wedge \mathit{finite})$
by (*meson FiniteImportSChopRight Prop04 lift-and-com*)

have *1*: $\vdash (\mathit{empty} \vee (\ \mathit{schopstar}\ f) \frown (f \wedge \mathit{finite})) \frown g = (\mathit{empty} \frown g \vee ((\mathit{schopstar}\ f) \frown f) \frown (g))$
using *01* **unfolding** *schop-d-def*
by (*metis 0 EmptyImpFinite OrChopEqv Prop10 SChopstar-finite inteq-reflection schop-d-def*)

have *11*: $\vdash \quad ((schopstar\ f)\ \frown\ f)\frown(g) = \ ((schopstar\ f))\frown(f\frown g)$
by (*meson Prop11 SChopAssoc*)
have *12*: $\vdash \ empty\frown g = g$
by (*simp add*: *EmptySChop*)
have *2*: $\vdash (schopstar\ f)\frown(f\ \frown g) = (schopstar\ f)\frown(\#False)$
by (*simp add*: *RightSChopEqvSChop assms*)
have *3*: $\vdash (schopstar\ f)\frown(\#False) = \#False$
by (*simp add*: *SChopRightFalse*)
show *?thesis*
by (*metis 0 1 11 12 3 assms int-simps*(*25*) *inteq-reflection*)
qed

lemma *WPowerstar-indepence2*:
assumes $\vdash f\ ;\ g = \#False$
shows $\vdash f\ ;\ (wpowerstar\ g) = f$
using *assms WPowerstar-inductr-var-eq2*[*of f g*] *WPowerstar-star2*[*of g*]
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty WPowerstar-inductr-var-equiv int-eq int-iffI int-simps*(*11*))

lemma *SChopstar-indepence2*:
assumes $\vdash f\ \frown\ g = \#False$
shows $\vdash f\ \frown\ (schopstar\ g) = (f \wedge finite)$
proof −
have *1*: $\vdash f\ \frown\ (schopstar\ g) = ((f \wedge finite) \vee (f\frown g)\frown(schopstar\ g))$
by (*metis ChopEmpty FPowerstarEqv FPowerstar-more-absorb SChopAssoc SChopOrEqv int-eq schop-d-def schopstar-d-def*)
have *2*: $\vdash (f\frown g)\frown(schopstar\ g) = \#False$
by (*metis AndInfChopEqvAndInf assms int-eq int-simps*(*19*) *schop-d-def*)
show *?thesis*
by (*metis 1 2 int-simps*(*25*) *inteq-reflection*)
qed

lemma *WPowerstar-lazy-comm*:
$(\vdash g;f \longrightarrow f;(wpowerstar\ (f \vee g)) \vee g) =$
$(\vdash g\ ;(wpowerstar\ f) \longrightarrow f\ ;\ (wpowerstar\ (f \vee g)) \vee g)$
proof
assume *1*: $\vdash g;f \longrightarrow f;(wpowerstar\ (f \vee g)) \vee g$
show $\vdash g\ ;(wpowerstar\ f) \longrightarrow f\ ;\ (wpowerstar\ (f \vee g)) \vee g$
proof −
have *2*: $\vdash (f;(wpowerstar\ (f \vee g)) \vee g);f =$
$((f;(wpowerstar\ (f \vee g)));f\ \vee g;f)$
by (*simp add*: *OrChopEqv*)
have *3*: $\vdash ((f;(wpowerstar\ (f \vee g)));f\ \vee g;f) \longrightarrow$
$((f;(wpowerstar\ (f \vee g)));f\ \vee (f;(wpowerstar\ (f \vee g))) \vee g)$
using *1* **by** *fastforce*
have *4*: $\vdash (f;(wpowerstar\ (f \vee g)));f \longrightarrow (f;(wpowerstar\ (f \vee g)))$
by (*metis ChopAssoc RightChopImpChop WPowerstar-subdist-var-1 WPowerstar-trans-eq inteq-reflection*)
have *5*: $\vdash ((f;(wpowerstar\ (f \vee g)));f\ \vee (f;(wpowerstar\ (f \vee g))) \vee g) \longrightarrow$

```
            ((f;(wpowerstar (f ∨ g))) ∨ g)
     using 4 by fastforce
   have 6: ⊢ g ∨ ((f;(wpowerstar (f ∨ g))) ∨ g) ⟶ ((f;(wpowerstar (f ∨ g))) ∨ g)
       by fastforce
   show ?thesis using WPowerstarInductR[of g LIFT (f;wpowerstar (f ∨ g) ∨ g) f]
    using 2 3 5 by fastforce
 qed
next
 assume 7: ⊢ g ;(wpowerstar f) ⟶ f ; (wpowerstar (f ∨ g)) ∨ g
 show  ⊢ g;f ⟶ f;(wpowerstar (f ∨ g)) ∨ g
 proof −
 have 80: ⊢ f;(wpowerstar f) = (f ∨ f;(wpowerstar f))
  by (meson ChopEmpty Prop02 Prop05 Prop11 RightChopImpChop WPowerstar-imp-empty lift-imp-trans)

  have 8: ⊢ (wpowerstar f) = (empty ∨ f ∨ f;(wpowerstar f))
   by (meson 80 Prop06 WPowerstarEqv)
  have 9: ⊢ g ;(wpowerstar f) = (g;empty ∨ g;f ∨ g;(f;(wpowerstar f)))
   by (metis 8 ChopOrEqv int-eq)
  show ?thesis
   using 7 9 by fastforce
 qed
qed

lemma SChopstar-lazy-comm:
(⊢ g⌢(f ∧ finite) ⟶ f⌢(schopstar (f ∨ g)) ∨ g) =
 (⊢ g ⌢(schopstar f) ⟶ f ⌢ (schopstar (f ∨ g)) ∨ g)
proof
 assume 1: ⊢ g⌢(f ∧ finite) ⟶ f⌢(schopstar (f ∨ g)) ∨ g
 show  ⊢ g ⌢(schopstar f) ⟶ f ⌢ (schopstar (f ∨ g)) ∨ g
  proof −
   have 2: ⊢ (f⌢(schopstar (f ∨ g)) ∨ g)⌢(f ∧ finite) =
            ((f⌢(schopstar (f ∨ g)))⌢(f ∧ finite)  ∨ g⌢(f ∧ finite))
    by (simp add: OrSChopEqv)
   have 3: ⊢ ((f⌢(schopstar (f ∨ g)))⌢(f ∧ finite)  ∨ g⌢(f∧ finite)) ⟶
            ((f⌢(schopstar (f ∨ g)))⌢(f ∧ finite)  ∨ (f⌢(schopstar (f ∨ g))) ∨ g)
      using 1
      using SChopAndA by fastforce
   have 4: ⊢ (f⌢(schopstar (f ∨ g)))⌢(f ∧ finite) ⟶ (f⌢(schopstar (f ∨ g)))
     using  SChopstar-subdist-var-1
    by (metis Prop11 RightSChopImpSChop SChopAssoc SChop-SChopstar-Closure SChopstardef lift-imp-trans)
   have 5: ⊢ ((f⌢(schopstar (f ∨ g)))⌢(f ∧ finite)  ∨ (f⌢(schopstar (f ∨ g))) ∨ g) ⟶
            ((f⌢(schopstar (f ∨ g))) ∨ g)
    using 4 by fastforce
   have 6: ⊢ g ∨ ((f⌢(schopstar (f ∨ g))) ∨ g) ⟶ ((f⌢(schopstar (f ∨ g))) ∨ g)
       by fastforce
   show ?thesis
   by (metis (mono-tags, lifting) 3 5 OrSChopEqv Prop03 SChopstar-inductr-var-equiv int-eq lift-imp-trans)

 qed
next
```

assume *7*: $\vdash g \frown (schopstar\ f) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$
show $\vdash g \frown (f \wedge finite) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$
proof −
have *8*: $\vdash (schopstar\ f) = (empty \vee (f \wedge finite) \vee f \frown (schopstar\ f))$
by (*meson AndSChopA Prop02 Prop05 Prop08 SChopstarMore-unfoldl-eq SChopstar-1L SChopstar-ext SChopstar-imp-empty int-iffD2 int-iffI*)
have *9*: $\vdash g \frown (schopstar\ f) = (g \frown empty \vee g \frown (f \wedge finite) \vee g \frown (f \frown (schopstar\ f)))$
by (*metis 8 SChopOrEqv inteq-reflection*)
show *?thesis*
using *7 9* **by** *fastforce*
qed
qed

3.5.6 WPowerstar

lemma *WPowerstar-and-inf*:
$\vdash (wpowerstar\ (f \wedge inf)) = (empty \vee (f \wedge inf))$
by (*simp add*: *AndInfChopEqvAndInf WPowerstar-boffa*)

lemma *WPowerstar-chop-and-finite-inf*:
$\vdash (wpowerstar\ f) = (wpowerstar\ (f \wedge finite)); (wpowerstar\ (f \wedge inf))$
by (*metis AndInfChopEqvAndInf OrFiniteInf WPowerstar-denest-var inteq-reflection*)

lemma *WPowerstar-SChopstar*:
$\vdash (wpowerstar\ f) = (schopstar\ f); (empty \vee (f \wedge inf))$
by (*metis SChopstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf int-eq*)

lemma *WPowerstar-empty*:
$\vdash (wpowerstar\ empty) = empty$
by (*simp add*: *WPowerstar-subid*)

lemma *WPowerstar-more*:
$\vdash (wpowerstar\ more)$
by (*metis ChopImpDi DiMoreEqvMore TrueW WPowerstar-boffa-var empty-d-def int-simps*(*29*) *inteq-reflection*)

lemma *WPowerstar-false*:
$\vdash (wpowerstar\ \#False) = empty$
by (*simp add*: *WPowerstar-subid*)

lemma *WPowerstar-true*:
$\vdash (wpowerstar\ \#True)$
by (*metis MP TrueW WPowerstar-ext*)

lemma *WPowerstar-inf*:
$\vdash (wpowerstar\ inf) = (empty \vee inf)$
by (*simp add*: *InfChopInfEqvInf WPowerstar-boffa*)

lemma *WPowerstar-finite*:

$\vdash$ (*wpowerstar finite*) = *finite*
by (*meson EmptyImpFinite FiniteChopFiniteEqvFinite Prop02 Prop11 WPowerstar-ext WPowerstar-induct-lvar-emp*

lemma *WPowerstar-imp-finite*:
assumes $\vdash$ $f \longrightarrow$ *finite*
shows $\vdash$ *wpowerstar* $f \longrightarrow$ *finite*
using *assms*
by (*metis WPowerstar-finite WPowerstar-iso inteq-reflection*)

lemma *WPowerstar-and-empty*:
$\vdash$ (*wpowerstar* ($f \wedge$ *empty*)) = *empty*
by (*metis AndEmptyChopAndEmptyEqvAndEmpty Prop11 Prop12 WPowerstar-subid inteq-reflection*)

lemma *WPowerstarIntro*:
assumes $\vdash$ $f \longrightarrow (g \wedge more); f$
shows $\vdash f \wedge$ *finite* $\longrightarrow$ *wpowerstar g*
proof −
have *14*: $\vdash$ *wpowerstar g* = (*empty* $\vee$ ($g \wedge$ *more*);(*wpowerstar g*))
using *WPowerstar-more-absorb*[*of g*] *WPowerstarEqv*[*of LIFT* ($g \wedge$ *more*)]
by (*metis int-eq*)
have *15*: $\vdash$ ($\neg$(*wpowerstar g*)) = (*more* $\wedge$ $\neg$(($g \wedge$ *more*);*wpowerstar g*))
using *14* **unfolding** *empty-d-def* **by** *fastforce*
have *20*: $\vdash f \wedge \neg$(*wpowerstar g*) $\longrightarrow$ ($g \wedge$ *more*);$f \wedge \neg$(($g \wedge$ *more*);*wpowerstar g*)
using *assms 15* **by** *fastforce*
have *21*: $\vdash g \wedge$ *more* $\longrightarrow$ *more*
by *auto*
from *20 21* **show** *?thesis* **using** *ChopContraB*[*of f LIFT wpowerstar g LIFT* ($g \wedge$ *more*)]
by *blast*
qed

lemma *WPowerstarIntroMore*:
assumes $\vdash$ $f \wedge$ *more*$\longrightarrow$ ($g \wedge$ *more*); *f*
shows $\vdash f \wedge$ *finite* $\longrightarrow$ *wpowerstar g*
proof −
have *14*: $\vdash$ *wpowerstar g* = (*empty* $\vee$ ($g \wedge$ *more*);(*wpowerstar g*))
using *WPowerstar-more-absorb*[*of g*] *WPowerstarEqv*[*of LIFT* ($g \wedge$ *more*)]
by (*metis int-eq*)
have *15*: $\vdash$ ($\neg$(*wpowerstar g*)) = (*more* $\wedge$ $\neg$(($g \wedge$ *more*);*wpowerstar g*))
using *14* **unfolding** *empty-d-def* **by** *fastforce*
have *20*: $\vdash f \wedge \neg$(*wpowerstar g*) $\longrightarrow$ ($g \wedge$ *more*);$f \wedge \neg$(($g \wedge$ *more*);*wpowerstar g*)
using *assms 15* **by** *fastforce*
have *21*: $\vdash g \wedge$ *more* $\longrightarrow$ *more*
by *auto*
from *20 21* **show** *?thesis* **using** *ChopContraB*[*of f LIFT wpowerstar g LIFT* ($g \wedge$ *more*)]
by *blast*
qed

lemma *Chopstar-WPowerstar*:
$\vdash$ *chopstar f* = *wpowerstar* (*f* $\wedge$ *more*)
by (*metis FPowerstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf chopstar-d-def inteq-reflection powerstar-d-def*)

lemma *Chopstar-FPowerstar*:
$\vdash$ *chopstar f* = (*fpowerstar f*);(*empty* $\vee$ (*f* $\wedge$ *inf*))
by (*metis Chopstar-WPowerstar FPowerstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf WPowerstar-more-absorb inteq-reflection*)

lemma *SChopstar-WPowerstar-more*:
$\vdash$ *schopstar f* = *wpowerstar* ((*f* $\wedge$ *more*) $\wedge$ *finite*)
by (*simp add*: *FPowerstar-WPowerstar schopstar-d-def*)

lemma *SChopstar-FPowerstar*:
$\vdash$ *schopstar f* = (*fpowerstar f*)
by (*simp add*: *FPowerstar-more-absorb schopstar-d-def*)

lemma *WPowerstar-skip-finite*:
$\vdash$ (*wpowerstar skip*) = *finite*
by (*metis EmptyImpFinite EmptyNextInducta Prop02 SkipChopFiniteImpFinite WPowerstar-1L WPowerstar-imp-empty WPowerstar-induct-lvar-empty int-iffI next-d-def*)

lemma *SPSEqvEmptyOrSChopSPS*:
$\vdash$ *fpowerstar f* = (*empty* $\vee$ *f*$\frown$*fpowerstar f*)
by (*simp add*: *FPowerstarEqv schop-d-def*)

lemma *ChopstarEqvPowerstar*:
$\vdash$ *f** = *powerstar f*
by (*metis Chopstar-FPowerstar powerstar-d-def*)

lemma *ChopplusCommute*:
$\vdash$ *f*;*f** = *f**;*f*
by (*metis Chopstar-WPowerstar WPowerstar-more-absorb WPowerstar-slide-var int-eq*)

lemma *SChopplusCommute*:
$\vdash$ *f*$\frown$*schopstar f* = *schopstar f*$\frown$(*f* $\wedge$ *finite*)
by (*meson Prop04 SChopstar-sim3 int-simps*(*27*))

lemma *CSEqvOrChopCSB*:
$\vdash$ *f** = (*empty* $\vee$ (*f**;*f*))
by (*metis ChopplusCommute ChopstarEqvPowerstar PowerstarEqvSem intI inteq-reflection*)

lemma *SCSEqvOrChopSCSB*:
$\vdash$ *schopstar f* = (*empty* $\vee$ (*schopstar f*$\frown$ (*f* $\wedge$ *finite*)))

by (*metis SChopplusCommute SChopstar-FPowerstar SPSEqvEmptyOrSChopSPS inteq-reflection*)

lemma *CSChopCS*:
$\vdash f^\star \,;\, f^\star = f^\star$
by (*metis Chopstar-WPowerstar WPowerstar-trans-eq inteq-reflection*)

lemma *SCSSChopSCS*:
$\vdash$ *schopstar f* $\frown$ *schopstar f* = *schopstar f*
by (*metis SChopstar-imp-empty SChopstar-invol SChopstar-sup-id-star1 inteq-reflection*)

lemma *CSCSEqvCS*:
$\vdash (f^\star)^\star \;= f^\star$
by (*metis* (*no-types*, *lifting*) *Chopstar-WPowerstar WPowerstar-invol WPowerstar-more-absorb int-eq*)

lemma *ChopPlusEqvOrChopChopPlus*:
$\vdash \quad (f;f^\star) \;= (f \vee \; f;\, (f;f^\star))$
by (*metis CSEqvOrChopCSB ChopEmpty ChopOrEqv ChopplusCommute inteq-reflection*)

lemma *SChopPlusEqvOrSChopSChopPlus*:
$\vdash \quad (f \frown$ *schopstar f*$) \;= ((f \wedge$ *finite*$) \vee \; f \frown (f \frown$ *schopstar f*$))$
by (*metis ChopEmpty SChopOrEqvRule SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *ChopPlusEqv*:
$\vdash (f;f^\star) \;= (f \vee \; (f \wedge \;$ *more* $);\, (f;f^\star))$
using *ChopplusCommute*[*of f*]
by (*metis ChopAssoc ChopstarEqv EmptyOrChopEqv int-eq*)

lemma *SChopPlusEqv*:
$\vdash (f \frown$ *schopstar f*$) \;= ((f \wedge$ *finite*$) \vee \; (f \wedge \;$ *more* $) \frown (f \frown$ *schopstar f*$))$
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc EmptyOrChopEqv Prop10 SChopstarEqv SChopplusCommute SChopstar-finite inteq-reflection schop-d-def*)

lemma *CSIntro*:
assumes $\vdash \quad f \longrightarrow (g \wedge \;$ *more* $);\, f$
shows $\vdash f \wedge$ *finite* $\longrightarrow g^\star$
by (*metis Chopstar-WPowerstar WPowerstarIntro WPowerstar-more-absorb assms inteq-reflection*)

lemma *CSIntroMore*:
assumes $\vdash \quad f \wedge$ *more* $\longrightarrow (g \wedge \;$ *more* $);\, f$
shows $\vdash f \wedge$ *finite* $\longrightarrow g^\star$
by (*metis Chopstar-WPowerstar WPowerstarIntroMore WPowerstar-more-absorb assms inteq-reflection*)

lemma *SCSIntro*:
assumes $\vdash \quad f \longrightarrow (g \wedge \;$ *more* $) \frown f$
shows $\vdash f \wedge$ *finite* $\longrightarrow$ *schopstar g*
proof −
have *1*: $\vdash ((g \wedge$ *more*$) \wedge$ *finite*$) = ((g \wedge$ *finite*$) \wedge$ *more*$)$

by *auto*
show *?thesis*
using *assms SChopstar-WPowerstar*[*of g*] *WPowerstarIntro*[*of f LIFT* (*g* ∧ *finite*)] *1*
by (*metis inteq-reflection schop-d-def*)
qed

lemma *SCSIntroMore*:
assumes ⊢ *f* ∧ *more* ⟶ (*g* ∧ *more*)⌢ *f*
shows ⊢ *f* ∧ *finite* ⟶ *schopstar g*
proof −
have *1*: ⊢((*g* ∧ *more*) ∧ *finite*) = ((*g* ∧ *finite*) ∧ *more*)
by *auto*
show *?thesis*
using *assms SChopstar-WPowerstar*[*of g*] *WPowerstarIntroMore*[*of f LIFT* (*g* ∧ *finite*)] *1*
by (*metis inteq-reflection schop-d-def*)
qed

lemma *CSElim*:
assumes ⊢ *empty* ⟶ *g*
⊢ (*f* ∧ *more*); *g* ⟶ *g*
shows ⊢ *f*⋆ ⟶ *g*
using *assms*
by (*metis Chopstar-WPowerstar Prop02 WPowerstar-induct-lvar-empty inteq-reflection*)

lemma *SCSElim*:
assumes ⊢ *empty* ⟶ *g*
⊢ (*f* ∧ *more*)⌢ *g* ⟶ *g*
shows ⊢ *schopstar f* ⟶ *g*
using *assms*
by (*metis Prop02 SChopstar-WPowerstar-more WPowerstar-induct-lvar-empty inteq-reflection schop-d-def*)

lemma *CSElimWithoutMore*:
assumes ⊢ *empty* ⟶ *g*
⊢ *f*; *g* ⟶ *g*
shows ⊢ *f*⋆ ⟶ *g*
using *assms*
by (*metis AndChopA CSElim Prop10 Prop12 int-eq*)

lemma *SCSElimWithoutMore*:
assumes ⊢ *empty* ⟶ *g*
⊢ *f*⌢ *g* ⟶ *g*
shows ⊢ *schopstar f* ⟶ *g*
using *assms*
by (*metis Prop02 SChopstar-WPowerstar WPowerstar-induct-lvar-empty inteq-reflection schop-d-def*)

lemma *WPowerstarChopEqvChopOrRule*:
assumes ⊢ *f*= ((*wpowerstar g*); *h*)
shows ⊢ *f* = ((*g*; *f*) ∨ *h*)
proof −

have *1*: $\vdash f = ((wpowerstar\ g);\ h)$ **using** *assms* **by** *auto*
have *2*: $\vdash (wpowerstar\ g) = (empty \vee (g;\ (wpowerstar\ g)))$ **by** (*simp add*: *WPowerstarEqv*)
hence *3*: $\vdash (wpowerstar\ g);\ h = (h \vee ((g;\ (wpowerstar\ g));\ h))$ **by** (*rule EmptyOrChopEqvRule*)
have *4*: $\vdash (g;\ (wpowerstar\ g));\ h = g;\ ((wpowerstar\ g);\ h)$ **by** (*meson ChopAssoc Prop11*)
hence *41*: $\vdash (wpowerstar\ g);\ h = (h \vee g;\ ((wpowerstar\ g);\ h))$ **using** *3* **by** *fastforce*
have *5*: $\vdash g;\ f = g;\ ((wpowerstar\ g);\ h)$ **using** *1* **by** (*rule RightChopEqvChop*)
hence *6*: $\vdash ((wpowerstar\ g);\ h) = (h \vee g;\ f)$ **using** *41* **by** *fastforce*
hence *61*: $\vdash ((wpowerstar\ g);\ h) = ((g;\ f) \vee h)$ **by** *auto*
from *1 61* **show** *?thesis* **by** *fastforce*
qed

lemma *CSChopEqvChopOrRule*:
assumes $\vdash f = (g^\star;\ h)$
shows $\vdash f = ((g;\ f) \vee h)$
using *assms*
by (*metis Chopstar-WPowerstar WPowerstarChopEqvChopOrRule WPowerstar-more-absorb int-eq*)

lemma *SCSChopEqvSChopOrRule*:
assumes $\vdash f = (schopstar\ g \frown h)$
shows $\vdash f = ((g \frown f) \vee h)$
using *assms*
by (*metis* (*no-types*, *lifting*) *FPowerstar-WPowerstar FPowerstar-more-absorb Prop10 SChopstar-finite WPowerstarChopEqvChopOrRule int-eq schop-d-def schopstar-d-def*)

lemma *WPowerstarChopIntroRule*:
assumes $\vdash f \wedge \neg h \longrightarrow g;\ f$
$\vdash g \longrightarrow more$
shows $\vdash f \wedge finite \longrightarrow (wpowerstar\ g);\ h$
proof −
have *1*: $\vdash f \wedge \neg h \longrightarrow g;\ f$
using *assms* **by** *blast*
have *2*: $\vdash g \longrightarrow more$
using *assms* **by** *blast*
hence *3*: $\vdash g \longrightarrow g \wedge more$
by *auto*
hence *4*: $\vdash g;\ f \longrightarrow (g\ );\ f$
by *auto*
have *5*: $\vdash f \longrightarrow (g\ );\ f \vee h$
using *1 4* **by** *fastforce*
have *6*: $\vdash wpowerstar\ g = (empty \vee g\ ;\ wpowerstar\ g)$
by (*simp add*: *WPowerstarEqv*)
hence *7*: $\vdash (wpowerstar\ g);\ h = (h \vee (g;\ wpowerstar\ g);\ h)$
by (*rule EmptyOrChopEqvRule*)
have *8*: $\vdash (g\ ;\ wpowerstar\ g);\ h = g\ ;\ (wpowerstar\ g;\ h)$
by (*meson ChopAssoc Prop11*)
have *9*: $\vdash (wpowerstar\ g);\ h = (h \vee g\ ;\ (wpowerstar\ g;\ h))$
using *7 8* **by** *fastforce*
have *10*: $\vdash f \wedge \neg\ (wpowerstar\ g;\ h) \longrightarrow (g);\ f \wedge \neg\ ((g);\ (wpowerstar\ g;\ h))$
using *5 9* **by** *fastforce*

have *11*: $\vdash g \wedge \ more \longrightarrow \ more$
by *fastforce*
from *10 11* **show** *?thesis* **using** *ChopContraB* **using** *2* **by** *blast*
qed

lemma *CSChopIntroRule*:
assumes $\vdash f \wedge \neg \ h \longrightarrow g; f$
$\vdash g \longrightarrow \ more$
shows $\vdash f \wedge$ *finite* $\longrightarrow g^{\star}; h$
using *assms*
by (*metis Chopstar-WPowerstar Prop10 WPowerstarChopIntroRule inteq-reflection*)

lemma *SCSChopIntroRule*:
assumes $\vdash \ f \wedge \neg \ h \longrightarrow g^{\frown} f$
$\vdash g \longrightarrow \ more$
shows $\vdash f \wedge$ *finite* $\longrightarrow$ *schopstar* $g^{\frown} h$
using *assms SChopstar-WPowerstar*[*of g*] *WPowerstarChopIntroRule*[*of f h LIFT* $(g \wedge$ *finite*$)$]
unfolding *schop-d-def*
by (*metis Prop05 Prop07 Prop10 SChopstar-finite finite-d-def inteq-reflection*)

lemma *DiamondAndEmptyEqvAndEmpty*:
$\vdash (\Diamond f \wedge empty) = (f \wedge empty)$
by (*metis ChopAndEmptyEqvEmptyChopEmpty EmptyChop FiniteAndEmptyEqvEmpty int-eq sometimes-d-def*)

lemma *InitAndEmptyEqvAndEmpty*:
$\vdash ((init\ w) \wedge empty) = (w \wedge empty)$
proof −
have *1*: $\vdash ((init\ w) \wedge empty) = ((w \wedge empty);\#True \wedge empty)$
by (*metis init-d-def int-eq lift-and-com*)
have *2*: $\vdash ((w \wedge empty);\#True \wedge empty) = (w \wedge empty);(\#True \wedge empty)$
by (*meson AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty lift-imp-trans Prop11 Prop12*)
have *3*: $\vdash (w \wedge empty);(\#True \wedge empty) = (w \wedge empty);empty$
using *RightChopEqvChop* **by** *fastforce*
have *4*: $\vdash (w \wedge empty);empty = (w \wedge empty)$
using *ChopEmpty* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *InitAndNotBoxInitImpNotEmpty*:
$\vdash init\ w \wedge \neg(\ \Box\ (init\ w)) \longrightarrow \neg \ empty$
proof −
have *1*: $\vdash ((init\ w) \wedge empty) = (w \wedge empty)$
by (*rule InitAndEmptyEqvAndEmpty*)
have *2*: $\vdash (\neg(\ \Box\ (init\ w)) \wedge empty) = (\Diamond\ (\neg\ (init\ w)) \wedge empty)$
by (*auto simp*: *always-d-def*)
have *3*: $\vdash (\Diamond\ (\neg\ (init\ w)) \wedge empty) = (\neg\ (init\ w) \wedge empty)$
by (*simp add*: *DiamondAndEmptyEqvAndEmpty*)

have *4*: $\vdash (\neg\ (init\ w)) = (init\ (\neg\ w))$
using *Initprop(2)* **by** *blast*
have *5*: $\vdash (\neg\ (init\ w) \wedge empty) = (\neg\ w \wedge empty)$
using *4 InitAndEmptyEqvAndEmpty* **by** (*metis inteq-reflection*)
have *6*: $\vdash (\neg(\ \Box\ (init\ w)) \wedge empty) = (\neg\ w \wedge empty)$
using *2 3 5* **by** *fastforce*
have *7*: $\vdash \neg(init\ w \wedge \neg(\ \Box\ (init\ w)) \wedge empty)$
using *1 6* **by** *fastforce*
from *7* **show** *?thesis* **by** *auto*
qed

lemma *BoxImpTrueChopAndEmpty*:
$\vdash \Box\ f \longrightarrow \#True;(f \wedge empty)$
using *BoxAndChopImport Finprop(3)* **by** *fastforce*

lemma *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*:
$\vdash \Box(\ init\ w) \wedge\ more\ \longrightarrow (\Box\ (init\ w) \wedge\ more\ ) \wedge\ fin\ (\ init\ w)$
proof −
have *1*: $\vdash fin\ (\ init\ w) = \#True\ ;\ (init\ w \wedge empty)$ **using** *FinEqvTrueChopAndEmpty* **by** *blast*
have *2*: $\vdash \Box(\ init\ w) \longrightarrow \#True;(init\ w \wedge empty)$ **by** (*rule BoxImpTrueChopAndEmpty*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpTrueSChopAndEmpty*:
$\vdash \Box\ f \wedge finite \longrightarrow \#True \frown (f \wedge empty)$
by (*metis BoxAndSChopImport DiamondEmptyEqvFinite TrueSChopEqvDiamond inteq-reflection*)

lemma *BoxInitAndMoreImpBoxInitAndMoreAndSFinInit*:
$\vdash \Box(\ init\ w) \wedge\ more\ \wedge finite \longrightarrow (\Box\ (init\ w) \wedge\ more\ ) \wedge\ sfin\ (\ init\ w)$
proof −
have *1*: $\vdash sfin\ (\ init\ w) = \#True \frown (init\ w \wedge empty)$ **using** *SFinEqvTrueSChopAndEmpty* **by** *blast*
have *2*: $\vdash \Box(\ init\ w) \wedge finite \longrightarrow \#True \frown (init\ w \wedge empty)$ **by** (*rule BoxImpTrueSChopAndEmpty*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *CSImpBox*:
assumes $\vdash f \longrightarrow\ empty\ \vee\ ((\Box\ (init\ w) \wedge\ more) \wedge finite\ );\ f$
shows $\vdash (init\ w \wedge f) \wedge finite \longrightarrow \Box\ (init\ w)$
proof −
have *1*: $\vdash f \longrightarrow\ empty\ \vee\ ((\Box(\ init\ w) \wedge\ more) \wedge finite\ );\ f$
using *assms* **by** *auto*
have *2*: $\vdash init\ w \wedge \neg(\ \Box\ (init\ w)) \longrightarrow \neg\ \ empty$
by (*rule InitAndNotBoxInitImpNotEmpty*)
have *3*: $\vdash init\ w \wedge f \wedge \neg(\ \Box\ (init\ w)) \longrightarrow ((\Box\ (init\ w) \wedge\ more) \wedge finite\ );\ f$
using *1 2* **by** *fastforce*
have *4*: $\vdash \Box\ (init\ w) \wedge\ more\ \longrightarrow (\Box\ (init\ w) \wedge\ more\ ) \wedge\ fin\ (\ init\ w)$
by (*rule BoxInitAndMoreImpBoxInitAndMoreAndFinInit*)
have *41*: $\vdash (\Box\ (init\ w) \wedge\ more) \wedge finite \longrightarrow$
$((\Box\ (init\ w) \wedge\ more\ ) \wedge finite) \wedge\ fin\ (\ init\ w)$
using *4* **by** *auto*

hence *5*: $\vdash ((\Box(\ init\ w) \wedge\ more) \wedge finite\ );\ f \longrightarrow$
$(((\Box\ (init\ w) \wedge\ more\ ) \wedge finite) \wedge\ fin\ (\ init\ w)\ );\ f$
by (*rule LeftChopImpChop*)
have *6*: $\vdash (((\Box\ (init\ w) \wedge\ more\ ) \wedge finite) \wedge\ fin\ (\ init\ w)\ );\ f =$
$((\Box(\ init\ w) \wedge\ more) \wedge finite\ );\ (init\ w \wedge f)$
using *AndFinChopEqvStateAndChop* **by** *blast*
have *7*: $\vdash \neg(\ \Box(\ init\ w)) \longrightarrow (\Box\ (init\ w))\ yields\ (\neg(\ \Box\ (init\ w)))$
by (*rule NotBoxStateImpBoxYieldsNotBox*)
have *8*: $\vdash (\Box(\ init\ w))\ yields\ (\neg\ (\Box\ (init\ w))) \longrightarrow$
$((\Box\ (init\ w) \wedge\ more) \wedge finite\ )\ yields\ (\neg(\ \Box(\ init\ w)))$
using *AndYieldsA*
by (*metis AndMoreAndFiniteEqvAndFmore inteq-reflection*)
have *9*: $\vdash ((\Box(\ init\ w) \wedge\ more) \wedge finite\ );\ (init\ w \wedge f) \wedge$
$((\Box(\ init\ w) \wedge\ more) \wedge finite\ )\ yields\ (\neg(\ \Box\ (init\ w)))$
$\longrightarrow$
$((\Box\ (init\ w) \wedge\ more) \wedge finite\ );\ ((init\ w \wedge f) \wedge \neg\ (\Box\ (init\ w)))$
by (*rule ChopAndYieldsImp*)
have *10*: $\vdash (init\ w \wedge f) \wedge \neg(\ \Box\ (init\ w)) \longrightarrow$
$((\Box(\ init\ w) \wedge\ more) \wedge finite\ );\ ((init\ w \wedge f) \wedge \neg(\ \Box\ (init\ w)))$
using *3 5 6 7 8 9* **by** *fastforce*
have *11*: $\vdash ((\Box(\ init\ w) \wedge\ more) \wedge finite\ );\ ((init\ w \wedge f) \wedge \neg(\ \Box\ (init\ w))) \longrightarrow$
$more\ ;\ ((init\ w \wedge f) \wedge \neg(\ \Box\ (init\ w))\ )$
by (*metis 41 LeftChopImpChop Prop12*)
have *12*: $\vdash (init\ w \wedge f) \wedge \neg(\ \Box\ (init\ w)) \longrightarrow$
$more\ ;\ ((init\ w \wedge f) \wedge \neg(\ \Box\ (init\ w))\ )$
using *10 11* **by** *fastforce*
from *12* **show** *?thesis* **using** *MoreChopContraFiniteB* **by** *blast*
qed

lemma *SCSImpBox*:
assumes $\vdash f \longrightarrow\ empty\ \vee\ ((\Box\ (init\ w) \wedge\ more)\ )^\frown f$
shows $\vdash (init\ w \wedge f) \wedge finite \longrightarrow \Box\ (init\ w)$
using *assms* **by** (*simp add*: *CSImpBox schop-d-def*)

lemma *ChopstarInductR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;(chopstar\ f) \longrightarrow h$
by (*metis Chopstar-WPowerstar WPowerstarInductR WPowerstar-more-absorb assms int-eq*)

lemma *BoxWPowerstarEqvBox*:
$\vdash\ \ (init\ w \wedge wpowerstar\ (\Box(\ init\ w))\ )\ = \Box\ (init\ w)$
proof −
have *1*: $\vdash \Box(\ init\ w);\Box(\ init\ w) \longrightarrow \Box(\ init\ w)$
by (*simp add*: *BoxStateChopBoxEqvBox int-iffD1*)
have *2*: $\vdash (init\ w \wedge empty) \longrightarrow \Box(\ init\ w)$
by (*simp add*: *StateAndEmptyImpBoxState*)
have *3*: $\vdash (init\ w \wedge empty) \vee \Box(\ init\ w);\Box(\ init\ w) \longrightarrow \Box(\ init\ w)$
using *1 2* **by** *fastforce*
have *4*: $\vdash (init\ w \wedge empty);wpowerstar\ (\Box(\ init\ w)) \longrightarrow \Box\ (init\ w)$

using *3 WPowerstarInductR* **by** *blast*
have *5*: $\vdash$ *init w* $\wedge$ *wpowerstar* ($\Box$(*init w*)) $\longrightarrow$ $\Box$ (*init w*)
using *4 StateAndEmptyChop* **by** *fastforce*
have *11*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ (*init w*)
using *BoxElim* **by** *blast*
have *12*: $\vdash$ $\Box$(*init w*) $\longrightarrow$ *wpowerstar* ($\Box$ (*init w*))
by (*simp add*: *WPowerstar-ext*)
have *13*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ *init w* $\wedge$ *wpowerstar* ($\Box$ (*init w*))
using *11 12* **by** *fastforce*
from *5 13* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxCSEqvBox*:
$\vdash$ (*init w* $\wedge$ ($\Box$(*init w*))*) = $\Box$ (*init w*)
by (*metis BoxWPowerstarEqvBox Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)

lemma *BoxSCSEqvBox*:
$\vdash$ (*init w* $\wedge$ *schopstar* ($\Box$(*init w*))) = ($\Box$ (*init w*) $\wedge$ *finite*)
proof −
have *1*: $\vdash$ $\Box$(*init w*)$\frown$($\Box$(*init w*) $\wedge$ *finite*)$\longrightarrow$ $\Box$(*init w*) $\wedge$ *finite*
by (*metis BoxStateAndChopEqvChop FiniteChopFiniteEqvFinite int-iffD2 inteq-reflection schop-d-def*)
have *2*: $\vdash$ (*init w* $\wedge$ *empty*) $\longrightarrow$ $\Box$(*init w*) $\wedge$ *finite*
using *EmptyImpFinite StateAndEmptyImpBoxState* **by** *fastforce*
have *3*: $\vdash$ (*init w* $\wedge$ *empty*) $\vee$ $\Box$(*init w*)$\frown$($\Box$(*init w*) $\wedge$ *finite*) $\longrightarrow$ $\Box$(*init w*) $\wedge$ *finite*
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*init w* $\wedge$ *empty*)$\frown$*schopstar* ($\Box$(*init w*)) $\longrightarrow$ $\Box$ (*init w*) $\wedge$ *finite*
using *SChopstarInductR 3*
by (*metis Prop12 SChopImpFinite SChopstar-finite SChopstar-finite-absorb inteq-reflection*)
have *5*: $\vdash$ *init w* $\wedge$ *schopstar* ($\Box$(*init w*)) $\longrightarrow$ $\Box$ (*init w*) $\wedge$ *finite*
using *4 StateAndEmptySChop* **by** *fastforce*
have *11*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ (*init w*)
using *BoxElim* **by** *blast*
have *12*: $\vdash$ $\Box$(*init w*) $\wedge$ *finite* $\longrightarrow$ *schopstar* ($\Box$ (*init w*))
by (*metis SChopstar-WPowerstar WPowerstar-ext inteq-reflection*)
have *13*: $\vdash$ $\Box$ (*init w*) $\wedge$ *finite* $\longrightarrow$ *init w* $\wedge$ *schopstar* ($\Box$ (*init w*))
using *11 12* **by** *fastforce*
from *5 13* **show** *?thesis* **by** *fastforce*
qed

lemma *WpowerstarAndMoreEqvAndMoreChop*:
$\vdash$ (*wpowerstar f* $\wedge$ *more*) = (*f* $\wedge$ *more*); *wpowerstar f*
proof −
have *1*: $\vdash$ (*empty* $\vee$ (*f* $\wedge$ *more*); *wpowerstar f*) $\wedge$ *more* $\longrightarrow$ (*f* $\wedge$ *more*); *wpowerstar f*
by (*auto simp*: *empty-d-def*)
have *2*: $\vdash$ *wpowerstar f* = (*empty* $\vee$ (*f* $\wedge$ *more*);*wpowerstar f*)
by (*metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have *3*: $\vdash$ *wpowerstar f* $\wedge$ *more* $\longrightarrow$ (*f* $\wedge$ *more*); *wpowerstar f*
using *1 2* **by** *fastforce*

have *4*: $\vdash (f \wedge \ more\ ); wpowerstar\ f \longrightarrow wpowerstar\ f$
using *2* **by** *fastforce*
have *5*: $\vdash (f \wedge \ more\ ) \longrightarrow more$
by *auto*
hence *6*: $\vdash (f \wedge \ more\ ); wpowerstar\ f \longrightarrow \ more$
by (*rule LeftChopImpMoreRule*)
have *7*: $\vdash (f \wedge \ more\ ); wpowerstar\ f \longrightarrow wpowerstar\ f \wedge \ more$
using *4 6* **by** *fastforce*
from *3 7* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreEqvAndMoreChop*:
$\vdash \ (f^\star \wedge \ more) \ = (f \wedge \ more\ ); f^\star$
by (*metis Chopstar-WPowerstar WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop int-eq*)

lemma *SCSAndMoreEqvAndMoreSChop*:
$\vdash \ (schopstar\ f \wedge \ more) \ = (f \wedge \ more\ ) \frown schopstar\ f$
proof −
have *1*: $\vdash((f \wedge more) \wedge finite) = ((f \wedge finite) \wedge more)$
by *auto*
show *?thesis*
using *SChopstar-WPowerstar*[*of f*] *WPowerstar-more-absorb*[*of LIFT* $(f \wedge finite)$]
WpowerstarAndMoreEqvAndMoreChop[*of LIFT* $(f \wedge finite)$]
by (*metis 1 int-eq schop-d-def*)
qed

lemma *SCSAndMoreEqvAndFMoreSChop*:
$\vdash \ (schopstar\ f \wedge \ fmore) \ = (f \wedge \ more\ ) \frown schopstar\ f$
by (*metis AndMoreAndFiniteEqvAndFmore Prop10 SCSAndMoreEqvAndMoreSChop SChopImpFinite SChop-star-finite*
inteq-reflection)

lemma *BoxStateAndWPowerstarEqvWPowerstar*:
$\vdash \ (\Box(\ init\ w) \wedge wpowerstar\ f \wedge finite) = (init\ w \wedge wpowerstar\ (\Box(\ init\ w) \wedge f) \wedge finite)$
proof −
have *1*: $\vdash \Box\ (init\ w) \longrightarrow init\ w$
using *BoxElim* **by** *blast*
have *2*: $\vdash (wpowerstar\ f \wedge \ more) \ = (f \wedge \ more\ ); wpowerstar\ f$
by (*simp add*: *WpowerstarAndMoreEqvAndMoreChop*)
have *3*: $\vdash (\Box(\ init\ w) \wedge ((f \wedge \ more\ ); wpowerstar\ f)) =$
$((\Box\ (init\ w) \wedge f \wedge \ more\ ); (\Box\ (init\ w) \wedge wpowerstar\ f))$
by (*rule BoxStateAndChopEqvChop*)
have *4*: $\vdash \Box\ (init\ w) \wedge f \wedge \ more \ \longrightarrow (\Box\ (init\ w) \wedge f) \wedge \ more$
by *auto*
hence *5*: $\vdash (\Box\ (init\ w) \wedge f \wedge \ more\ ); (\Box\ (init\ w) \wedge wpowerstar\ f) \longrightarrow$
$((\Box\ (init\ w) \wedge f) \wedge \ more\ ); (\Box\ (init\ w) \wedge wpowerstar\ f)$
by (*rule LeftChopImpChop*)
have *6*: $\vdash (\Box(\ init\ w) \wedge wpowerstar\ f) \wedge \ more \ \longrightarrow$

$((\Box\ (init\ w) \wedge f) \wedge\ \ more\ );\ (\Box\ (init\ w) \wedge wpowerstar\ f)$
using *2 3 5* **by** *fastforce*
hence *7*: $\vdash (\Box\ (init\ w) \wedge wpowerstar\ f) \wedge finite \longrightarrow wpowerstar\ (\Box\ (init\ w) \wedge f)$
using *WPowerstarIntroMore*[*of LIFT* $(\Box\ (init\ w) \wedge wpowerstar\ f)$ *LIFT* $(\Box\ (init\ w) \wedge f)$]
by *blast*
have *71*: $\vdash init\ w \wedge \Box\ (init\ w) \wedge wpowerstar\ f \wedge finite \longrightarrow init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \wedge finite$
using *7* **by** *fastforce*
have *8*: $\vdash \Box(\ init\ w) \wedge wpowerstar\ f \wedge finite \longrightarrow init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \wedge finite$
using *1 71* **by** *fastforce*
have *11*: $\vdash wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow wpowerstar\ (\Box\ (init\ w))$
by (*meson Prop12 WPowerstar-iso int-iffD2 lift-and-com*)
have *12*: $\vdash (init\ w \wedge wpowerstar\ (\Box\ (init\ w))) = \Box\ (init\ w)$
by (*simp add*: *BoxWPowerstarEqvBox*)
have *13*: $\vdash wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow wpowerstar\ f$
by (*meson Prop12 WPowerstar-iso int-iffD2 lift-and-com*)
have *14*: $\vdash init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow init\ w \wedge wpowerstar\ (\Box\ (init\ w)) \wedge wpowerstar\ f$
using *11 13* **by** *fastforce*
have *15*: $\vdash init\ w \wedge wpowerstar\ (\Box\ (init\ w)) \wedge wpowerstar\ f \longrightarrow \Box\ (init\ w) \wedge wpowerstar\ f$
using *12* **by** *auto*
have *16*: $\vdash init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow \Box\ (init\ w) \wedge wpowerstar\ f$
using *14 15 lift-imp-trans* **by** *blast*
from *8 16* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateAndCSEqvCS*:
$\vdash\ \ (\Box(\ init\ w) \wedge f^\star \wedge finite) = (init\ w \wedge (\Box(\ init\ w) \wedge f)^\star \wedge finite)$
by (*metis BoxStateAndWPowerstarEqvWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *BoxStateAndSCSEqvSCS*:
$\vdash\ \ (\Box(\ init\ w) \wedge schopstar\ f) = (init\ w \wedge schopstar\ (\Box(\ init\ w) \wedge f))$
proof −
have *1*: $\vdash (\Box\ (init\ w) \wedge f \wedge finite) = ((\Box\ (init\ w) \wedge f) \wedge finite)$
by *auto*
show *?thesis*
using *BoxStateAndWPowerstarEqvWPowerstar*[*of w LIFT* $(f \wedge finite)$]
SChopstar-WPowerstar[*of f*] *SChopstar-WPowerstar*[*of LIFT* $(\Box(\ init\ w) \wedge f)$]
SChopstar-finite[*of f*] *SChopstar-finite*[*of LIFT* $(\Box(\ init\ w) \wedge f)$]
by (*metis 1 Prop10 inteq-reflection*)
qed

lemma *BaWPowerstarImpWPowerstar*:
$\vdash\ \ \ ba\ (f \longrightarrow g) \wedge finite \longrightarrow wpowerstar\ f \longrightarrow wpowerstar\ g$
proof −
have *1*: $\vdash wpowerstar\ f =\ \ (empty\ \ \vee\ \ (f \wedge\ \ more\ );\ wpowerstar\ f)$
by (*metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have *2*: $\vdash wpowerstar\ g =\ \ (empty\ \ \vee\ \ (g \wedge\ \ more\ );\ wpowerstar\ g)$
by (*metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have *21*: $\vdash \neg(wpowerstar\ g) =\ \ (\neg empty\ \ \wedge \neg(\ (g \wedge\ \ more\ );\ wpowerstar\ g))$
using *2* **by** *fastforce*

hence *22*: $\vdash \neg(wpowerstar\ g) = (more\ \wedge \neg(\ (g \wedge\ more\ );\ wpowerstar\ g))$
by (*simp add*: *empty-d-def*)
have *3*: $\vdash wpowerstar\ f \wedge \neg\ (wpowerstar\ g) \longrightarrow$
$(empty\ \vee (f \wedge\ more\ );\ wpowerstar\ f) \wedge more \wedge \neg\ ((g \wedge\ more\ );\ wpowerstar\ g)$
using *1 22* **by** *fastforce*
have *31*: $\vdash ((empty\ \vee (f \wedge\ more\ );\ wpowerstar\ f) \wedge more) = ((f \wedge\ more\ );\ wpowerstar\ f \wedge more)$
by (*auto simp*: *empty-d-def*)
have *32*: $\vdash wpowerstar\ f \wedge \neg\ (wpowerstar\ g) \longrightarrow (f \wedge\ more\ );\ wpowerstar\ f \wedge \neg\ ((g \wedge\ more\ );\ wpowerstar\ g)$
using *3 31* **by** *fastforce*
have *4*: $\vdash (f \longrightarrow g) \longrightarrow (f \wedge\ more\ \longrightarrow g \wedge\ more\ )$
by *auto*
hence *5*: $\vdash\ ba\ (f \longrightarrow g) \longrightarrow\ ba\ (f \wedge\ more\ \longrightarrow g \wedge\ more\ )$
by (*rule BaImpBa*)
have *6*: $\vdash\ ba\ (f \wedge\ more\ \longrightarrow g \wedge\ more\ ) \longrightarrow$
$(f \wedge\ more\ );\ wpowerstar\ f \longrightarrow\ (g \wedge\ more\ );\ wpowerstar\ f$
by (*rule BaLeftChopImpChop*)
have *7*: $\vdash\ ba\ (f \longrightarrow g) \wedge (f \wedge more\ );\ wpowerstar\ f \longrightarrow (g \wedge\ more\ );\ wpowerstar\ f$
using *5 6* **by** *fastforce*
have *8*: $\vdash (g \wedge\ more\ );\ wpowerstar\ f \wedge \neg\ ((g \wedge\ more\ );\ wpowerstar\ g)$
$\longrightarrow\ (g \wedge\ more\ );\ (wpowerstar\ f \wedge \neg\ (wpowerstar\ g))$
by (*rule ChopAndNotChopImp*)
have *9*: $\vdash (g \wedge\ more\ );\ (wpowerstar\ f \wedge \neg\ (wpowerstar\ g)) \longrightarrow\ more\ ;\ (wpowerstar\ f \wedge \neg\ (wpowerstar\ g))$
by (*rule AndChopB*)
have *10*: $\vdash\ ba\ (f \longrightarrow g) \longrightarrow more\ ;\ (wpowerstar\ f \wedge \neg\ (wpowerstar\ g)) \longrightarrow$
$more\ ;\ (\ ba\ (f \longrightarrow g) \wedge wpowerstar\ f \wedge \neg\ (wpowerstar\ g))$
by (*rule BaChopImpChopBa*)
have *11*: $\vdash\ ba\ (f \longrightarrow g) \wedge wpowerstar\ f \wedge \neg\ (wpowerstar\ g) \longrightarrow$
$more\ ;\ (\ ba\ (f \longrightarrow g) \wedge wpowerstar\ f \wedge \neg\ (wpowerstar\ g))$
using *32 7 8 9 10* **by** *fastforce*
hence *12*: $\vdash finite \longrightarrow \neg\ (\ (ba\ (f \longrightarrow g)) \wedge (wpowerstar\ f) \wedge (\neg\ (wpowerstar\ g)))$
using *MoreChopLoopFiniteB* **by** *blast*
from *12* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *BaCSImpCS*:
$\vdash\ \ ba\ (f \longrightarrow g) \wedge finite \longrightarrow f^\star \longrightarrow g^\star$
by (*metis BaWPowerstarImpWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SDa-Da*:
$\vdash sda\ f = da\ (f \wedge finite)$
unfolding *sda-d-def da-d-def schop-d-def*
by *simp*

lemma *SBa-Ba*:
$\vdash sba\ f = ba\ (finite \longrightarrow f)$
proof −

have *1*: $\vdash (\neg\ (\mathit{finite} \longrightarrow f)) = (\ \mathit{finite} \wedge \neg\ f\ )$
by *auto*
show *?thesis*
unfolding *sba-d-def ba-d-def sda-d-def da-d-def schop-d-def*
by (*metis 1 AndChopCommute int-eq int-simps*(*17*))
qed

lemma *SBaSCSImpSCS*:
$\vdash \quad \mathit{sba}\ (f \longrightarrow g) \longrightarrow \mathit{schopstar}\ f \longrightarrow \mathit{schopstar}\ g$
proof −
have *1*: $\vdash \mathit{ba}\ (\mathit{finite} \longrightarrow f \longrightarrow g) \longrightarrow \mathit{ba}\ (f \wedge \mathit{finite} \longrightarrow g \wedge \mathit{finite})$
by (*simp add*: *BaImpBa intI*)
have *2*: $\vdash \mathit{schopstar}\ f \longrightarrow \mathit{finite}$
by (*simp add*: *SChopstar-finite*)
show *?thesis*
using *BaWPowerstarImpWPowerstar*[*of LIFT* $(f \wedge \mathit{finite})$ *LIFT* $(g \wedge \mathit{finite})$]
SChopstar-WPowerstar[*of f*] *SChopstar-WPowerstar*[*of g*]
SBa-Ba[*of LIFT* $(f \longrightarrow g)$] *1 2* **by** *fastforce*
qed

lemma *BaWPowerstarEqvWPowerstar*:
$\vdash \quad \mathit{ba}\ (f = g) \wedge \mathit{finite} \longrightarrow (\mathit{wpowerstar}\ f = \mathit{wpowerstar}\ g)$
proof −
have *0*: $\vdash (f = g) = (\ (f \longrightarrow g) \wedge\ \ (g \longrightarrow f))$
by *fastforce*
have *1*: $\vdash \mathit{ba}\ (f = g) = (\mathit{ba}\ (f \longrightarrow g) \wedge \mathit{ba}\ (g \longrightarrow f))$
by (*metis 0 BaAndEqv int-eq*)
have *2*: $\vdash \mathit{ba}\ (f \longrightarrow g) \wedge \mathit{finite} \longrightarrow (\mathit{wpowerstar}\ f \longrightarrow \mathit{wpowerstar}\ g)$
by (*rule BaWPowerstarImpWPowerstar*)
have *3*: $\vdash \mathit{ba}\ (g \longrightarrow f) \wedge \mathit{finite} \longrightarrow (\mathit{wpowerstar}\ g \longrightarrow \mathit{wpowerstar}\ f)$
by (*rule BaWPowerstarImpWPowerstar*)
have *4*: $\vdash \mathit{ba}\ (f = g) \wedge \mathit{finite} \longrightarrow (\mathit{wpowerstar}\ f \longrightarrow \mathit{wpowerstar}\ g) \wedge\ \ (\mathit{wpowerstar}\ g \longrightarrow \mathit{wpowerstar}\ f)$
using *1 2 3* **by** *fastforce*
have *5*: $\vdash ((\mathit{wpowerstar}\ f \longrightarrow \mathit{wpowerstar}\ g) \wedge (\mathit{wpowerstar}\ g \longrightarrow \mathit{wpowerstar}\ f)) = (\mathit{wpowerstar}\ f = \mathit{wpowerstar}\ g)$
by *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaCSEqvCS*:
$\vdash \quad \mathit{ba}\ (f = g) \wedge \mathit{finite} \longrightarrow (f^\star = g^\star)$
by (*metis BaWPowerstarEqvWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SBaSCSEqvSCS*:
$\vdash \quad \mathit{sba}\ (f = g) \longrightarrow (\mathit{schopstar}\ f = \mathit{schopstar}\ g)$
proof −
have *0*: $\vdash (f = g) = (\ (f \longrightarrow g) \wedge (g \longrightarrow f))$
by *fastforce*

have *1*: $\vdash$ *sba* $(f = g) = (sba\ (f \longrightarrow g) \wedge sba\ (g \longrightarrow f))$
by (*metis 0 SBaAndEqv int-eq*)
show *?thesis* **using** *SBaSCSImpSCS*[*of f g*] *SBaSCSImpSCS*[*of g f*]
1 **by** *fastforce*
qed

lemma *BaAndWPowerstarImport*:
$\vdash$ *ba* $f \wedge wpowerstar\ g \wedge finite \longrightarrow wpowerstar\ (f \wedge g)$
proof −
have *1*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence *2*: $\vdash$ *ba* $f \longrightarrow$ *ba* $(g \longrightarrow f \wedge g)$ **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba* $(g \longrightarrow f \wedge g) \wedge finite \longrightarrow wpowerstar\ g \longrightarrow wpowerstar\ (f \wedge g)$
by (*rule BaWPowerstarImpWPowerstar*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndCSImport*:
$\vdash$ *ba* $f \wedge g^\star \wedge finite \longrightarrow (f \wedge g)^\star$
by (*metis BaAndWPowerstarImport Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SBaAndSCSImport*:
$\vdash$ *sba* $f \wedge schopstar\ g \longrightarrow schopstar\ (f \wedge g)$
proof −
have *1*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence *2*: $\vdash$ *sba* $f \longrightarrow$ *sba* $(g \longrightarrow f \wedge g)$ **by** (*rule SBaImpSBa*)
have *3*: $\vdash$ *sba* $(g \longrightarrow f \wedge g) \longrightarrow schopstar\ g \longrightarrow schopstar\ (f \wedge g)$ **by** (*rule SBaSCSImpSCS*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

3.5.7 Len

lemma *wpower-len*:
$\vdash wpower\ skip\ k = len\ k$
by (*simp add*: *len-d-def*)

lemma *wpowerstar-skip-finite*:
$\vdash finite = wpowerstar\ skip$
using *WPowerstar-skip-finite* **by** *fastforce*

lemma *schopstar-skip-finite*:
$\vdash finite = schopstar\ skip$
by (*metis Prop10 SChopstar-WPowerstar WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)

lemma *wpowerstar-skip-fmore*:
$\vdash fmore = skip;wpowerstar\ skip$
by (*metis FmoreEqvSkipChopFinite inteq-reflection wpowerstar-skip-finite*)

lemma *schopstar-skip-fmore*:
$\vdash fmore = skip^\frown schopstar\ skip$
by (*metis NextSChopdef WPowerstar-skip-finite inteq-reflection next-d-def schopstar-skip-finite*

wpowerstar-skip-fmore)

lemma *len-k-finite*:
$\vdash$ *len k* $\longrightarrow$ *finite*
proof (*induct k*)
case *0*
then show *?case*
by (*metis EmptyImpFinite len-d-def wpow-0*)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: $\vdash$ *len* (*Suc k*) = *skip*;*len k*
by (*simp add*: *len-d-def*)
have *2*: $\vdash$ *skip* $\longrightarrow$ *finite*
by (*metis WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
show *?thesis*
by (*metis 1 2 ChopImpChop FiniteChopFiniteEqvFinite Suc inteq-reflection*)
qed
qed

lemma *len-k-schop*:
$\vdash$ *len Suc k* = *len k* $\frown$ *skip*
unfolding *len-d-def* **unfolding** *schop-d-def*
by (*metis Prop10 WPowerCommute int-eq len-k-finite wpow-Suc wpower-len*)

lemma *SkipChopAnd*:
$\vdash$ ((*skip*;*f*) $\wedge$ (*skip*;*g*)) = *skip*;(*f* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *skip*;(*f* $\wedge$ *g*) $\longrightarrow$ ((*skip*;*f*) $\wedge$ (*skip*;*g*))
by (*simp add*: *ChopAndA ChopAndB Prop12*)
have *2*: $\vdash$ ((*skip*;*f*) $\wedge$ (*skip*;*g*)) $\longrightarrow$ *skip*;(*f* $\wedge$ *g*)
by (*metis NextAndEqvNextAndNext SkipChopEqvNext int-eq int-iffD2*)
show *?thesis*
using *1 2 int-iffI* **by** *blast*
qed

lemma *SkipSChopAnd*:
$\vdash$ ((*skip*$\frown$*f*) $\wedge$ (*skip*$\frown$*g*)) = *skip*$\frown$(*f* $\wedge$ *g*)
by (*metis NextAndNextEqvNextRule NextSChopdef inteq-reflection lift-and-com*)

lemma *LenChopAnd*:
$\vdash$ (*len k*;*f* $\wedge$ *len k*;*g*) = *len k*;(*f* $\wedge$ *g*)
proof −
have *2*: $\vdash$ *len k*;(*f* $\wedge$ *g*) $\longrightarrow$ (*len k*;*f* $\wedge$ *len k*;*g*)
by (*simp add*: *ChopAndA ChopAndB Prop12*)
have *1*: $\vdash$ (*len k*;*f* $\wedge$ *len k*;*g*) $\longrightarrow$ *len k*;(*f* $\wedge$ *g*)
proof (*induct k arbitrary*: *f g*)
case *0*

then show *?case*
by (*metis EmptyChop int-iffD2 inteq-reflection wpow-0 wpower-len*)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: ⊢ *len Suc k;f = len k;(skip;f)*
using *WPowerCommute*[*of LIFT skip k*] *ChopAssoc*[*of LIFT len k LIFT skip f*]
by (*metis inteq-reflection wpow-Suc wpower-len*)
have *2*: ⊢ *len Suc k;g = len k;(skip;g)*
using *WPowerCommute*[*of LIFT skip k*] *ChopAssoc*[*of LIFT len k LIFT skip g*]
by (*metis inteq-reflection wpow-Suc wpower-len*)
have *3*: ⊢ (*len k;(skip;f)* ∧ *len k;(skip;g)*) ⟶ *len k;((skip;f)* ∧ *(skip;g))*
by (*simp add*: *Suc*)
have *4*: ⊢ ((*skip;f)* ∧ *(skip;g)*) = *skip;(f* ∧ *g)*
by (*simp add*: *SkipChopAnd*)
have *5*: ⊢ *len k;((skip;f)* ∧ *(skip;g))* = *len* (*Suc k*)*;(f* ∧ *g)*
using *WPowerCommute*[*of LIFT skip k*] *ChopAssoc*[*of LIFT len k LIFT skip LIFT* (*f* ∧ *g*)]
by (*metis 4 inteq-reflection wpow-Suc wpower-len*)
show *?thesis*
by (*metis 1 2 3 5 int-eq*)
qed
qed
show *?thesis*
using *1 2 int-iffI* **by** *blast*
qed

lemma *LenSChopAnd*:
⊢ (*len k⌢f* ∧ *len k⌢g*) = *len k⌢*(*f* ∧ *g*)
proof −
have *1*: ⊢ *len k* ⟶ *finite*
using *len-k-finite* **by** *blast*
show *?thesis* **unfolding** *schop-d-def*
by (*metis 1 LenChopAnd Prop10 int-eq*)
qed

lemma *LenEqvLenChopLen*:
⊢ *len*(*i*+*j*) = *len*(*i*);*len*(*j*)
proof
(*induct i*)
case *0*
then show *?case*
by (*metis EmptyChop Prop11 add-cancel-left-left len-d-def wpow-0*)
next
case (*Suc i*)
then show *?case*
by (*metis ChopAssoc add-Suc int-eq len-d-def wpow-Suc*)
qed

lemma *LenChopFalse*:

$\vdash$ *len k* ;*#False* $\longrightarrow$ *#False*
by (*metis AndInfEqvChopFalse ChopImpChop InfEqvNotFinite int-iffD1 int-simps*(*21*) *inteq-reflection len-k-finite*)

lemma *LenSChopFalse*:
$\vdash$ *len k* $\frown$*#False* $\longrightarrow$ *#False*
by (*metis AndChopB InfEqvNotFinite TrueChopAndFiniteEqvAndFiniteChopFinite infinite-d-def int-eq int-simps*(*19*) *int-simps*(*22*) *schop-d-def*)

lemma *len-len-suc-not*:
$\vdash \neg$(*len k* $\wedge$ *len* (*Suc k*);*f*)
proof −
have *1*: $\vdash$ *len k* = *len k*;*empty*
by (*meson ChopEmpty Prop11*)
have *2*: $\vdash$ *len* (*Suc k*);*f* = *len k*;(*skip*;*f*)
using *ChopAssoc*[*of LIFT len k LIFT skip f*] *WPowerCommute*[*of LIFT skip k*]
by (*metis inteq-reflection wpow-Suc wpower-len*)
have *3*: $\vdash$ (*len k*;*empty* $\wedge$ *len k*;(*skip*;*f*)) = *len k*;(*empty* $\wedge$ *skip*;*f*)
by (*simp add*: *LenChopAnd*)
have *4*: $\vdash$ (*empty* $\wedge$ *skip*;*f*) $\longrightarrow$ *#False*
unfolding *empty-d-def more-d-def next-d-def*
by (*metis ChopAndB Prop01 int-simps*(*16*) *int-simps*(*25*) *int-simps*(*4*) *inteq-reflection*)
have *5*: $\vdash$ (*len k* $\wedge$ *len* (*Suc k*);*f*) $\longrightarrow$ *len k*;*#False*
by (*metis 1 2 3 4 RightChopImpChop inteq-reflection*)
have *6*: $\vdash$ *len k* ;*#False* $\longrightarrow$ *#False*
using *LenChopFalse* **by** *blast*
show *?thesis*
by (*metis 5 6 int-simps*(*14*) *inteq-reflection lift-imp-trans*)
qed

lemma *len-len-suc-not-schop*:
$\vdash \neg$(*len k* $\wedge$ *len* (*Suc k*)$\frown$*f*)
unfolding *schop-d-def*
by (*metis AndInfEqvChopFalse LenChopFalse Prop09 Prop10 int-eq int-simps*(*14*) *itl-def*(*8*) *len-len-suc-not*)

lemma *Finite-exist-len*:
$\vdash$ *finite* = ($\exists$ *k*. *len k*)
by (*metis ExEqvRule WPowerstar-skip-finite int-eq wpower-len wpowerstar-d-def*)

lemma *LenNPlusOneB*:
$\vdash$ *len*(*n*+*1*) = *len*(*n*);*skip*
proof −
have *1*: $\vdash$ *len*(*n*+*1*) = *len*(*n*);*len*(*1*) **by** (*rule LenEqvLenChopLen*)
have *2*: $\vdash$ *len*(*1*) = *skip* **by** (*simp add*: *ChopEmpty len-d-def*)
hence *3*: $\vdash$ *len*(*n*);*len*(*1*) = *len*(*n*);*skip* **using** *RightChopEqvChop* **by** *blast*
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LenCommute*:
$\vdash$ (*skip*;(*len n*)) = (*len n*);*skip*

by (*simp add*: *WPowerCommute len-d-def*)

lemma *NotFixedChop*:
$\vdash (\neg((g \wedge len(k));f)) = (\neg(di(g \wedge len(k))) \vee ((g \wedge len(k));(\neg f)))$
by (*auto simp add*: *itl-defs len-defs Valid-def min-def nlength-eq-enat-nfiniteD*)

3.5.8 Properties of While

lemma *SChopstar-Chopstar*:
$\vdash$ *schopstar* f = *chopstar* $(f \wedge finite)$
by (*metis Chopstar-WPowerstar SChopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)

lemma *SWhile-While*:
$\vdash$ *swhile* g *do* f = *while* g *do* $(f \wedge finite)$
proof −
have *1*: $\vdash ((g \wedge f) \wedge finite) = (g \wedge f \wedge finite)$
by *auto*
have *2*: $\vdash$ *chopstar* $((g \wedge f) \wedge finite)$ = *chopstar* $(g \wedge f \wedge finite)$
using *1* **by** (*metis Chopstardef int-eq*)
show *?thesis*
unfolding *swhile-d-def while-d-def*
using *SFinEQvFinAndFinite*[*of LIFT* $\neg$ g] *SChopstar-Chopstar*[*of LIFT* $(g \wedge f)$]
2 SChopstar-finite[*of LIFT* $(g \wedge f)$] **by** *fastforce*
qed

lemma *IfAndFiniteDist*:
$\vdash$ $(if_i$ *(init w) then* $(f;g)$ *else empty* $\wedge$ *finite*) =
$(if_i$ *(init w) then* $((f \wedge finite);(g \wedge finite))$ *else empty*)
proof −
have *1*: $\vdash$ $(if_i$ *(init w) then* $(f;g)$ *else empty* $\wedge$ *finite*) =
$((\ (init\ w \wedge (f;g)) \vee (\neg(init\ w) \wedge empty)) \wedge finite)$
by (*auto simp*: *ifthenelse-d-def*)
have *2*: $\vdash ((\ (init\ w \wedge (f;g)) \vee (\neg(init\ w) \wedge empty)) \wedge finite) =$
$((\ (init\ w \wedge (f;g) \wedge finite) \vee (\neg(init\ w) \wedge empty \wedge finite)))$
by *auto*
have *3*: $\vdash (init\ w \wedge (f;g) \wedge finite) = (init\ w \wedge (f \wedge finite);(g \wedge finite))$
using *ChopAndFiniteDist* **by** *fastforce*
have *4*: $\vdash (\neg(init\ w) \wedge empty \wedge finite) = (\neg(init\ w) \wedge empty)$
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have *5*: $\vdash ((\ (init\ w \wedge (f;g) \wedge finite) \vee (\neg(init\ w) \wedge empty \wedge finite))) =$
$((init\ w \wedge (f \wedge finite);(g \wedge finite)) \vee (\neg(init\ w) \wedge empty))$
by (*metis 2 3 4 inteq-reflection*)
have *6*: $\vdash ((init\ w \wedge (f \wedge finite);(g \wedge finite)) \vee (\neg(init\ w) \wedge empty)) =$
$(if_i$ *(init w) then* $((f \wedge finite);(g \wedge finite))$ *else empty*)
by (*auto simp*: *ifthenelse-d-def*)
from *1 2 5 6* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *WhileEqvIf*:

$\vdash ((\mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ f) \wedge \mathit{finite}) =$
$\quad (\ \mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ (f;\ (\ \mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ f))\ \ \mathit{else}\ \ \ \mathit{empty}\ \wedge \mathit{finite})$

proof −

have *1*: $\vdash\ (\mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ f \wedge \mathit{finite}) =$
$\quad ((((\mathit{init}\ w) \wedge f)^\star \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w))) \wedge \mathit{finite})$
by (*simp add*: *while-d-def*)

have *2*: $\vdash (\mathit{init}\ w \wedge f)^\star =\ \ (\mathit{empty}\ \vee\ \ ((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star))$
by (*metis CSEqvOrChopCSB ChopplusCommute int-eq*)

have *21*: $\vdash (((\mathit{init}\ w) \wedge f)^\star \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}) =$
$\quad ((\mathit{empty}\ \vee\ \ ((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star)) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite})$
using *2* **by** *fastforce*

have *22*: $\vdash ((\mathit{empty}\ \vee\ \ ((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star)) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}) =$
$\quad ((\ \mathit{empty} \wedge \mathit{fin}\ (\neg(\mathit{init}\ w)) \wedge \mathit{finite}) \vee$
$\quad (\ ((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}))$
by *auto*

have *23*: $\vdash ((((\mathit{init}\ w) \wedge f)^\star \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w))) \wedge \mathit{finite}) =$
$\quad ((\ \mathit{empty} \wedge \mathit{fin}\ (\neg(\mathit{init}\ w)) \wedge \mathit{finite}) \vee$
$\quad (\ ((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}))$
using *21 22* **by** *auto*

have *3*: $\vdash\ \ (\mathit{empty}\ \ \wedge\ \ \mathit{fin}\ (\neg\ (\ \mathit{init}\ w))) = (\neg\ (\ \mathit{init}\ w) \wedge\ \ \mathit{empty})$
by (*metis FinAndEmpty Prop04 lift-and-com*)

hence *31*: $\vdash\ \ (\mathit{empty}\ \ \wedge\ \ \mathit{fin}\ (\neg\ (\ \mathit{init}\ w)) \wedge \mathit{finite}) = (\neg\ (\ \mathit{init}\ w) \wedge\ \ \mathit{empty} \wedge \mathit{finite})$
by *auto*

have *32*: $\vdash (\neg\ (\ \mathit{init}\ w) \wedge\ \ \mathit{empty} \wedge \mathit{finite}) = (\neg\ (\ \mathit{init}\ w) \wedge\ \ \mathit{empty})$
using *FiniteAndEmptyEqvEmpty* **by** *auto*

have *33*: $\vdash (\ \mathit{empty} \wedge \mathit{fin}\ (\neg(\mathit{init}\ w)) \wedge \mathit{finite}) = (\neg\ (\ \mathit{init}\ w) \wedge\ \ \mathit{empty})$
using *31 32* **by** *fastforce*

have *34*: $\vdash ((\ \mathit{empty} \wedge \mathit{fin}\ (\neg(\mathit{init}\ w)) \wedge \mathit{finite}) \vee$
$\quad (\ ((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite})) =$
$\quad ((\neg\ (\ \mathit{init}\ w) \wedge\ \ \mathit{empty}) \vee$
$\quad (\ ((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}))$
using *23 33* **by** *auto*

have *4*: $\vdash (\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star = (\mathit{init}\ w \wedge (f;\ (\mathit{init}\ w \wedge f)^\star))$
by (*rule StateAndChop*)

have *41*: $\vdash (((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}) =$
$\quad (\mathit{init}\ w \wedge (f;\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite})$
using *4* **by** *auto*

have *42*: $\vdash (\mathit{init}\ w \wedge ((f;\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite})) =$
$\quad (\mathit{init}\ w \wedge ((f;\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ \ \ (\mathit{init}\ (\neg\ w)) \wedge \mathit{finite}))$
using *Initprop*(*2*) **by** (*metis StateAndEmptyChop int-eq*)

have *5*: $\vdash ((f;\ ((\mathit{init}\ w \wedge f)^\star)) \wedge\ \ (\mathit{fin}\ \ (\ \mathit{init}\ (\neg\ w))) \wedge \mathit{finite})$
$\quad = ((f \wedge \mathit{finite});\ ((\mathit{init}\ w \wedge f)^\star \wedge\ \ (\mathit{fin}\ \ (\ \mathit{init}\ (\neg\ w))) \wedge \mathit{finite}))$
using *ChopAndFin* **by** *fastforce*

hence *49*: $\vdash (\mathit{init}\ w \wedge (f;\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}\ ) =$
$\quad (\mathit{init}\ \ w \wedge (f \wedge \mathit{finite});\ ((\mathit{init}\ w \wedge f)^\star \wedge\ \ (\mathit{fin}\ \ (\ \mathit{init}\ (\neg\ w))) \wedge \mathit{finite}))$
using *42* **by** *fastforce*

have *50*: $\vdash\ \ (((\mathit{init}\ w \wedge f);\ (\mathit{init}\ w \wedge f)^\star) \wedge\ \ \mathit{fin}\ (\neg\ \ (\mathit{init}\ w)) \wedge \mathit{finite}) =$
$\quad (\mathit{init}\ \ w \wedge (f \wedge \mathit{finite});\ ((\mathit{init}\ w \wedge f)^\star \wedge\ \ (\mathit{fin}\ \ (\ \mathit{init}\ (\neg\ w))) \wedge \mathit{finite}))$
by (*meson 41 49 Prop04 lift-and-com*)

have *51*: $\vdash (\mathit{init}\ \ w \wedge (f \wedge \mathit{finite});\ ((\mathit{init}\ w \wedge f)^\star \wedge\ \ (\mathit{fin}\ \ (\ \mathit{init}\ (\neg\ w))) \wedge \mathit{finite}\ )) =$

$(init\ \ w \wedge (f \wedge finite);\ ((init\ w \wedge f)^\star \wedge\ \ (fin\ (\neg\ (\ init\ \ w))) \wedge finite))$
by (*metis* (*no-types*) *EmptyChop Initprop*(*2*) *inteq-reflection*)
have *52*: $\vdash$ $(((init\ w \wedge f);\ (init\ w \wedge f)^\star) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite) =$
$(init\ w \wedge ((f \wedge finite);\ ((init\ w \wedge f)^\star \wedge\ \ fin\ (\neg\ (\ init\ \ w)) \wedge finite))\ )$
using *50 51* **by** *fastforce*
have *53*: $\vdash$ $((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$(\ ((init\ w \wedge f);\ (init\ w \wedge f)^\star) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite)) =$
$((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$(init\ w \wedge ((f \wedge finite);\ ((init\ w \wedge f)^\star \wedge\ \ fin\ (\neg\ (\ init\ \ w)) \wedge finite))\ ))$
using *52 34* **by** *auto*
have *6*: $\vdash$ $((f \wedge finite);\ (((init\ w \wedge f)^\star \wedge\ \ fin\ (\neg\ (\ init\ w))) \wedge finite)) =$
$(f \wedge finite);\ \ (while\ \ (init\ w)\ \ do\ \ f \wedge finite)$
by (*simp add*: *while-d-def*)
have *61*: $\vdash$ $(init\ w \wedge ((f \wedge finite);\ (((init\ w \wedge f)^\star \wedge\ \ fin\ (\neg\ (\ init\ \ w))) \wedge finite))) =$
$(init\ w \wedge\ \ ((f \wedge finite);\ \ (while\ \ (init\ w)\ \ do\ \ f \wedge finite)))$
using *6* **by** *auto*
have *62*: $\vdash$ $((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$(init\ w \wedge ((f \wedge finite);\ (((init\ w \wedge f)^\star \wedge\ \ fin\ (\neg\ (\ init\ \ w))) \wedge finite))\ ))$
$=((\neg\ (\ init\ w) \wedge\ \ empty\ ) \vee$
$(init\ w \wedge ((f \wedge finite);(while\ \ (init\ w)\ \ do\ \ f \wedge finite))\ )\ )$
using *61* **by** *fastforce*
have *7*: $\vdash$ $(while\ \ (init\ w)\ \ do\ \ f\ \ \wedge finite)$
$= (((\neg\ (\ init\ w) \wedge\ \ empty\ ) \vee$
$(init\ w \wedge (f\ ;\ \ while\ \ (init\ w)\ \ do\ \ f) \wedge finite)))$
proof −
have $\vdash$$(while\ (init\ w)\ do\ f \wedge finite) =$
$(\neg\ init\ w \wedge empty \vee (init\ w \wedge f);(init\ w \wedge f)^\star \wedge fin\ (\neg\ init\ w) \wedge finite)$
by (*metis 1 23 34 inteq-reflection*)
then show *?thesis*
by (*metis 21 22 34 52 ChopAndFiniteDist int-eq*)
qed
have *71*: $\vdash$ $((if_i\ \ (init\ w)\ then\ (f;\ (\ while\ \ (init\ w)\ \ do\ \ f))\ \ else\ \ \ empty)\ \ \wedge finite)=$
$(((\neg\ (\ init\ w) \wedge\ \ empty\ ) \vee\ \ (init\ w \wedge (f;\ \ while\ \ (init\ w)\ \ do\ \ f)) \wedge finite\ ))$
using *FiniteAndEmptyEqvEmpty* **by** (*auto simp*: *ifthenelse-d-def*)
from *7 71* **show** *?thesis* **by** *fastforce*
qed

lemma *WPower-import-finite*:
$\vdash$ $(wpower\ f\ k \wedge finite) = wpower\ (f \wedge finite)\ k$
proof (*induct k*)
case *0*
then show *?case*
using *FiniteAndEmptyEqvEmpty* **by** *fastforce*
next
case (*Suc k*)
then show *?case*
by (*metis ChopAndFiniteDist inteq-reflection wpow-Suc*)
qed

lemma *WPowerstar-import-finite*:
$\vdash$ (*wpowerstar f* $\wedge$ *finite*) = (*wpowerstar* (*f* $\wedge$ *finite*))
proof −
have *1*: $\vdash$ (*wpowerstar* (*f* $\wedge$ *finite*)) $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*)
by (*metis OrFiniteInf Prop12 SChopstar-finite SChopstar-WPowerstar WPowerstar-subdist inteq-reflection*)
have *2*: $\vdash$ *wpowerstar f* $\wedge$ *finite* $\longrightarrow$ (*wpowerstar* (*f* $\wedge$ *finite*))
unfolding *wpowerstar-d-def* **using** *WPower-import-finite*[*of f*] **by** *fastforce*
show *?thesis*
using *1 2 int-iffI* **by** *blast*
qed

lemma *Chopstar-import-finite*:
$\vdash$ (*chopstar f* $\wedge$ *finite*) = (*chopstar* (*f* $\wedge$ *finite*))
using *Chopstar-WPowerstar*[*of f*] *Chopstar-WPowerstar*[*of LIFT* (*f* $\wedge$ *finite*)]
by (*metis WPowerstar-import-finite WPowerstar-more-absorb int-eq*)

lemma *AndMoreSChopAndMoreEqvAndMoreSChop*:
$\vdash$ ((*f* $\wedge$ *more*)$\frown$*g* $\wedge$ *more*) = (*f* $\wedge$ *more*)$\frown$*g*
by (*meson AndSChopB MoreSChopImpMore Prop10 Prop11 lift-imp-trans*)

lemma *WPowerstar-chopstar*:
$\vdash$ (*wpowerstar* (*f* $\wedge$ *more*)) = (*chopstar f*)
by (*meson Chopstar-WPowerstar Prop11*)

lemma *While-import-finite*:
$\vdash$ (*while g do f* $\wedge$ *finite*) = (*while g do* (*f* $\wedge$ *finite*))
proof −
have *1*: $\vdash$(*g* $\wedge$ *f* $\wedge$ *finite*) = ((*g* $\wedge$ *f*) $\wedge$ *finite*)
by *auto*
have *2*: $\vdash$ *chopstar* (*g* $\wedge$ *f* $\wedge$ *finite*) = *chopstar* ((*g* $\wedge$ *f*) $\wedge$ *finite*)
by (*metis 1 Chopstardef int-eq*)
show *?thesis*
unfolding *while-d-def*
using *Chopstar-import-finite*[*of LIFT* (*g* $\wedge$ *f*)] *2* **by** *fastforce*
qed

lemma *SWhileEqvIf*:
$\vdash$ *swhile* (*init w*) *do f* = *if*$_i$ (*init w*) *then* (*f*$\frown$ (*swhile* (*init w*) *do f*)) *else empty*
proof −
have *1*: $\vdash$ ((*init w* $\wedge$ *f*) $\wedge$ *finite*) = (*init w* $\wedge$ *f* $\wedge$ *finite*)
by *auto*
have *2*: $\vdash$ *chopstar* ((*init w* $\wedge$ *f*) $\wedge$ *finite*) = *chopstar* (*init w* $\wedge$ *f* $\wedge$ *finite*)
using *1* **by** (*metis Chopstardef inteq-reflection*)
have *3*: $\vdash$ ((*while* (*init w*) *do f*) $\wedge$ *finite*) = *while* (*init w*) *do* (*f* $\wedge$ *finite*)
unfolding *while-d-def*
using *Chopstar-import-finite*[*of LIFT* (*init w* $\wedge$ *f*)] *2* **by** *fastforce*

have *4*: $\vdash$ *while* (*init w*) *do* (*f* $\wedge$ *finite*) = *swhile* (*init w*) *do* *f*
by (*metis Prop04 SWhile-While lift-and-com swhile-d-def*)
have *5*: $\vdash$ ((*init w* $\wedge$ (*f* $\wedge$ *finite*);*while* (*init w*) *do* (*f* $\wedge$ *finite*) $\vee \neg$ *init w* $\wedge$ *empty*) $\wedge$ *finite*) =
((*init w* $\wedge$ (*f* $\wedge$ *finite*);*while* (*init w*) *do* (*f* $\wedge$ *finite*) $\wedge$ *finite* $\vee \neg$ *init w* $\wedge$ *empty* $\wedge$ *finite*))
by *fastforce*
have *6*: $\vdash$ ((*f* $\wedge$ *finite*);*while* (*init w*) *do* (*f* $\wedge$ *finite*) $\wedge$ *finite*) =
(*f* $\wedge$ *finite*);*while* (*init w*) *do* (*f* $\wedge$ *finite*)
by (*metis 3 ChopAndFiniteDist Prop10 Prop12 int-eq int-iffD2*)
have *7*: $\vdash$ ($\neg$ *init w* $\wedge$ *empty* $\wedge$ *finite*) = ($\neg$ *init w* $\wedge$ *empty*)
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have *8*: $\vdash$ (*if*$_i$ (*init w*) *then* ((*f* $\wedge$ *finite*); (*while* (*init w*) *do* (*f* $\wedge$ *finite*))) *else* *empty* $\wedge$ *finite*) =
(*if*$_i$ (*init w*) *then* (*f*$\frown$ (*swhile* (*init w*) *do* *f*)) *else* *empty*)
unfolding *ifthenelse-d-def schop-d-def*
by (*metis 4 5 6 7 inteq-reflection*)
have *9*: $\vdash$ ((*while* (*init w*) *do* (*f* $\wedge$ *finite*)) $\wedge$ *finite*) =
(*if*$_i$ (*init w*) *then* ((*f* $\wedge$ *finite*); (*while* (*init w*) *do* (*f* $\wedge$ *finite*))) *else* *empty* $\wedge$ *finite*)
by (*simp add*: *WhileEqvIf*)
show *?thesis*
by (*metis 3 4 8 9 Prop10 Prop12 int-iffD2 inteq-reflection*)
qed

lemma *WhileChopEqvIf*:
$\vdash$ ((*while* (*init w*) *do f*) $\wedge$ *finite*); *g* =
if$_i$ (*init w*) *then* ((*f* $\wedge$ *finite*); (((*while* (*init w*) *do f*) $\wedge$ *finite*) ; *g*)) *else* *g*
proof $-$
have *1*: $\vdash$ (*while* (*init w*) *do* *f* $\wedge$ *finite*)=
(*if*$_i$ (*init w*) *then* (*f*; (*while* (*init w*) *do* *f*)) *else* *empty* $\wedge$ *finite*)
by (*rule WhileEqvIf*)
have *11*: $\vdash$ (*if*$_i$ (*init w*) *then* (*f*; (*while* (*init w*) *do* *f*)) *else* *empty* $\wedge$ *finite*) =
(*if*$_i$ (*init w*) *then* ((*f* $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*)) *else* *empty*)
using *IfAndFiniteDist* **by** *fastforce*
have *12*: $\vdash$ (*while* (*init w*) *do* *f* $\wedge$ *finite*) =
(*if*$_i$ (*init w*) *then* ((*f* $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*)) *else* *empty*)
using *1 11* **by** *fastforce*
hence *2*: $\vdash$ ((*while* (*init w*) *do* *f*) $\wedge$ *finite*); *g* =
(*if*$_i$ (*init w*)
then (((*f* $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*));*g*)
else (*empty*;*g*))
by (*rule IfChopEqvRule*)
have *3*: $\vdash$ *empty* ; *g* = *g*
by (*rule EmptyChop*)
have *4*: $\vdash$ (*if*$_i$ (*init w*)
then (((*f* $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*));*g*)
else (*empty*;*g*)) =
(*if*$_i$ (*init w*)
then (((*f* $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*));*g*)
else *g*)
using *3* **using** *inteq-reflection* **by** *fastforce*

have *5*: $\vdash$ (((*f* $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*));*g*) =
((*f* $\wedge$ *finite*);((*while* (*init w*) *do* *f* $\wedge$ *finite*);*g*))
by (*meson ChopAssoc Prop11*)
have *6*: $\vdash$ (*if*$_i$ (*init w*)
then (((*f* $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*));*g*)
else *g*) =
(*if*$_i$ (*init w*)
then ((*f* $\wedge$ *finite*);((*while* (*init w*) *do* *f* $\wedge$ *finite*);*g*))
else *g*)

using *5* **using** *inteq-reflection* **by** *fastforce*
show *?thesis*
using *2 4 6* **by** *fastforce*
qed

lemma *SWhileSChopEqvIf*:
$\vdash$ (*swhile* (*init w*) *do f*)$^\frown$*g* = *if*$_i$ (*init w*) *then* (*f*$^\frown$((*swhile* (*init w*) *do f*)$^\frown$ *g*)) *else* *g*
unfolding *schop-d-def*
by (*metis* (*no-types*, *opaque-lifting*) *ChopEmpty EmptySChop SChopAssoc SWhile-While WhileChopEqvIf*
inteq-reflection schop-d-def)

lemma *WhileChopEqvIfRule*:
assumes $\vdash$ *f* = (*while* (*init w*) *do* *g* $\wedge$ *finite*); *h*
shows $\vdash$ *f* = *if*$_i$ (*init w*) *then* ((*g* $\wedge$ *finite*); *f*) *else* *h*
proof −
have *1*: $\vdash$ *f* = (*while* (*init w*) *do* *g* $\wedge$ *finite*); *h*
using *assms* **by** *auto*
have *2*: $\vdash$ (*while* (*init w*) *do* *g* $\wedge$ *finite*); *h* =
if$_i$ (*init w*) *then* ((*g* $\wedge$ *finite*); ((*while* (*init w*) *do* *g* $\wedge$ *finite*); *h*)) *else* *h*
by (*rule WhileChopEqvIf*)
have *3*: $\vdash$ ((*g* $\wedge$ *finite*); *f*) = ((*g* $\wedge$ *finite*); ((*while* (*init w*) *do* *g* $\wedge$ *finite*); *h*))
using *1* **by** (*rule RightChopEqvChop*)
have *4*: $\vdash$ ((*g* $\wedge$ *finite*); ((*while* (*init w*) *do* *g* $\wedge$ *finite*); *h*)) = ((*g* $\wedge$ *finite*); *f*)
using *3* **by** *auto*
have *5*: $\vdash$ *if*$_i$ (*init w*) *then* ((*g* $\wedge$ *finite*); ((*while* (*init w*) *do* *g* $\wedge$ *finite*); *h*)) *else* *h* =
if$_i$ (*init w*) *then* ((*g* $\wedge$ *finite*); *f*) *else* *h*
using *4* **using** *inteq-reflection* **by** *fastforce*
from *1 2 5* **show** *?thesis* **by** *fastforce*
qed

lemma *SWhileSChopEqvIfRule*:
assumes $\vdash$ *f* = (*swhile* (*init w*) *do* *g*)$^\frown$ *h*
shows $\vdash$ *f* = *if*$_i$ (*init w*) *then* (*g*$^\frown$ *f*) *else* *h*
using *assms*
by (*metis SWhileSChopEqvIf inteq-reflection*)

lemma *WhileImpFin*:
$\vdash$ *while* (*init w*) *do* *f* $\longrightarrow$ *fin* ($\neg$ (*init w*))
proof −

have *1*: $\vdash$ $(init\ w \wedge f)^\star \wedge\ fin\ (\neg\ (\ init\ w)) \longrightarrow\ fin\ (\neg\ (\ init\ w))$ **by** *auto*
from *1* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *SWhileImpFin*:
$\vdash$ $\ swhile\ \ (init\ w)\ \ do\ \ f \longrightarrow\ \ sfin\ (\neg\ (\ init\ w))$
by (*simp add*: *Prop01 Prop05 swhile-d-def*)

lemma *WhileEqvEmptyOrChopWhile*:
$\vdash$ $(while\ (init\ w)\ do\ f \wedge finite)=$
$((\neg\ (init\ w) \wedge empty) \vee (init\ w \wedge ((f \wedge more) \wedge finite);(while\ (init\ w)\ do\ f \wedge finite)))$
proof −
have *1*: $\vdash$ $(init\ w \wedge f)^\star\ =\ (empty\ \vee\ ((init\ w \wedge f) \wedge more\ );\ (init\ w \wedge f)^\star)$
by (*rule ChopstarEqv*)
have *2*: $\vdash$ $((init\ w \wedge f) \wedge\ \ more)\ = (init\ w \wedge (f \wedge more))$
by *auto*
hence *3*: $\vdash$ $((init\ w \wedge f) \wedge\ \ more);\ (init\ w \wedge f)^\star\ = (init\ w \wedge f \wedge more);\ (init\ w \wedge f)^\star$
by (*rule LeftChopEqvChop*)
have *4*: $\vdash$ $(init\ w \wedge f)^\star\ =\ (empty\ \vee\ \ (init\ w \wedge f \wedge\ \ more\ );\ (init\ w \wedge f)^\star)$
using *1 3* **by** *fastforce*
have *40*: $\vdash$ $fin\ (\neg\ (\ init\ w)) = fin\ (\neg\ w)$
by (*metis FinEqvTrueChopAndEmpty InitAndEmptyEqvAndEmpty Initprop(2) inteq-reflection*)
have *5*: $\vdash$ $((init\ w \wedge f)^\star \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ =$
$((\ empty\ \wedge\ \ fin\ \ (\neg\ (init\ w)) \wedge finite) \vee$
$((init\ w \wedge f \wedge\ \ more\ );\ (init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ w)) \wedge finite))$
using *1 4* **by** *fastforce*
have *51*: $\vdash$ $((empty\ \wedge\ \ fin\ \ (\neg\ (\ init\ w))) \wedge finite) = ((\neg\ (\ init\ w) \wedge\ \ empty) \wedge finite)$
by (*metis FinAndEmpty inteq-reflection lift-and-com*)
have *52*:$\vdash$ $((\neg\ (\ init\ w) \wedge\ \ empty) \wedge finite) = (\neg\ (\ init\ w) \wedge\ \ empty)$
using *EmptyImpFinite* **by** *auto*
have *6*: $\vdash$ $((empty\ \wedge\ \ fin\ \ (\neg\ (\ init\ w))) \wedge finite) = (\neg\ (\ init\ w) \wedge\ \ empty)$
using *51 52* **by** *fastforce*
have *60*: $\vdash$ $((\ empty\ \wedge\ \ fin\ \ (\neg\ (init\ w)) \wedge finite) \vee$
$((init\ w \wedge f \wedge\ \ more\ );\ (init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ w)) \wedge finite)) =$
$((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$((init\ w \wedge f \wedge\ \ more\ );\ (init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ w)) \wedge finite))$
using *6* **by** *fastforce*
have *61*: $\vdash$ $((init\ w \wedge f)^\star \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ =$
$((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$((init\ w \wedge f \wedge\ \ more\ );\ (init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ w)) \wedge finite))$
by (*metis 5 60 inteq-reflection*)
have *70*: $\vdash$ $(init\ w \wedge f \wedge\ \ more\ );\ (init\ w \wedge f)^\star = (init\ w \wedge (f \wedge\ \ more\ );\ (init\ w \wedge f)^\star)$
by (*rule StateAndChop*)
have *7*: $\vdash$ $((init\ w \wedge f \wedge\ \ more\ );\ (init\ w \wedge f)^\star \wedge fin\ \ (\ init\ (\neg\ w)) \wedge finite) =$
$(init\ w \wedge (f \wedge\ \ more\ );\ (init\ w \wedge f)^\star \wedge fin\ \ (\ init\ (\neg\ w)) \wedge finite)$
using *70* **by** *auto*
have *8*: $\vdash$ $((\ (f \wedge\ \ more\ );\ (init\ w \wedge f)^\star) \wedge\ \ fin\ \ (\ init\ (\neg\ w)) \wedge finite)\ =$
$(((f \wedge\ \ more) \wedge finite);\ ((init\ w \wedge f)^\star \wedge\ \ fin\ (\ \ init\ (\neg\ w)) \wedge finite))$
using *ChopAndFin* **by** *fastforce*

have *81*: $\vdash$ *fin* (*init* ($\neg$ *w*)) = *fin* ($\neg$ (*init w*))
by (*meson FinEqvFin Initprop*(*2*) *Prop11*)
have *82*: $\vdash$ ((*f* $\wedge$ *more*); (*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(((*f* $\wedge$ *more*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*))
using *8 81* **by** (*metis inteq-reflection*)
have *83*: $\vdash$ (*init w* $\wedge$ (*f* $\wedge$ *more*); (*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*))
using *82* **by** *fastforce*
have *84*: $\vdash$ (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ (*f* $\wedge$ *more*); (*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$(*init* (*w*))) $\wedge$ *finite*)) =
(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)))
using *82* **by** *auto*
have *9*: $\vdash$ ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);
((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)))
by (*metis 61 7 81 84 inteq-reflection*)
have *10*: $\vdash$ ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*))) $\wedge$ *finite*)
by *auto*
hence *11*: $\vdash$ (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);
((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*))) =
(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);
(((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*))) $\wedge$ *finite*)))
by (*metis EmptyChop int-eq*)
have *12*: $\vdash$ ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);
(((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*))) $\wedge$ *finite*)))
by (*metis 11 9 inteq-reflection*)
from *12* **show** *?thesis*
by (*metis 10 inteq-reflection while-d-def*)
qed

lemma *SWhileEqvEmptyOrSChopSWhile*:
$\vdash$ *swhile* (*init w*) *do f* = (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$ (*init w* $\wedge$ (*f* $\wedge$ *more*)$\frown$*swhile* (*init w*) *do f*))
proof −
have *1*: $\vdash$ (((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*) = ((*f* $\wedge$ *more*) $\wedge$ *finite*)
by *auto*
show *?thesis*
unfolding *schop-d-def*
using *WhileEqvEmptyOrChopWhile*[*of w LIFT* (*f*)] *SWhile-While*[*of LIFT* (*init w*) *f*]
by (*metis While-import-finite inteq-reflection*)
qed

lemma *WhileIntro*:
assumes $\vdash$ $\neg$ (*init w*) $\wedge$ *f* $\longrightarrow$ *empty*

$\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); *f*
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *while* (*init w*) *do g*
proof −
have *1*: $\vdash$ ¬ (*init w*) $\wedge$ *f* $\longrightarrow$ *empty*
using *assms* **by** *blast*
have *2*: $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); *f*
using *assms* **by** *blast*
have *3*: $\vdash$ (*while* (*init w*) *do g* $\wedge$ *finite*)=
((¬ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))
by (*rule WhileEqvEmptyOrChopWhile*)
hence *31*: $\vdash$ ¬ (*while* (*init w*) *do g* $\wedge$ *finite*) =
(¬((¬ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))))
by *fastforce*
hence *32*: $\vdash$ (*f* $\wedge$ ¬ (*while* (*init w*) *do g* $\wedge$ *finite*)) =
(*f* $\wedge$ ¬((¬(*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))))
by *fastforce*
have *33*: $\vdash$ (*f* $\wedge$ ¬((¬(*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))) =
(*f* $\wedge$ ¬(¬(*init w*) $\wedge$ *empty*) $\wedge$
¬(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))
by *auto*
have *34*: $\vdash$ (*f* $\wedge$ ¬(¬(*init w*) $\wedge$ *empty*) $\wedge$
¬((*init w*) $\wedge$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))) =
(*f* $\wedge$ ((*init w*) $\vee$ *more*) $\wedge$
(¬(*init w*) $\vee$ ¬(((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))))
by (*auto simp*: *empty-d-def*)
have *35*: $\vdash$ (*f* $\wedge$ ((*init w*) $\vee$ *more*) $\wedge$
(¬(*init w*) $\vee$ ¬(((*g* $\wedge$ *more*) $\wedge$ *finite*);(*while* (*init w*) *do g* $\wedge$ *finite*)))) =
((*f* $\wedge$ (*init w*) $\wedge$ ¬ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ (*init w*) $\wedge$ ¬(*init w*)) $\vee$
(*f* $\wedge$ *more* $\wedge$ ¬ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ *more* $\wedge$ ¬(*init w*)))
by *auto*
have *36*: $\vdash$ (*f* $\wedge$ ¬ (*while* (*init w*) *do g* $\wedge$ *finite*)) =
((*f* $\wedge$ (*init w*) $\wedge$ ¬ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ (*init w*) $\wedge$ ¬(*init w*)) $\vee$
(*f* $\wedge$ *more* $\wedge$ ¬ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ *more* $\wedge$ ¬(*init w*)))
by (*metis 32 33 34 35 int-eq*)
have *37*: $\vdash$ ¬(*f* $\wedge$ *more* $\wedge$ ¬(*init w*))
using *1* **by** (*auto simp*: *empty-d-def*)
have *38*: $\vdash$ (*f* $\wedge$ *more* $\wedge$ ¬ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\longrightarrow$
(((*g* $\wedge$ *more*) $\wedge$ *finite*); *f* $\wedge$
¬ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))
using *1 2* **by** (*auto simp*: *empty-d-def Valid-def*)
have *39*: $\vdash$ (*f* $\wedge$ (*init w*) $\wedge$ ¬ (((*g* $\wedge$ *more*)$\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\longrightarrow$
(((*g* $\wedge$ *more*) $\wedge$ *finite*); *f* $\wedge$

$\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *g* $\wedge$ *finite*)))
using *2* **by** *auto*
have *40*: $\vdash$ ((*f* $\wedge$ (*init w*) $\wedge$ $\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ (*init w*) $\wedge$ $\neg$(*init w*)) $\vee$
(*f* $\wedge$ *more* $\wedge$ $\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ *more* $\wedge$ $\neg$(*init w*))) $\longrightarrow$
((*g* $\wedge$ *more*) $\wedge$ *finite*); *f* $\wedge$
$\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *g* $\wedge$ *finite*))
using *39 38 37 38* **by** *fastforce*
have *4*: $\vdash$ *f* $\wedge$ $\neg$ (*while* (*init w*) *do* *g* $\wedge$ *finite*) $\longrightarrow$
((*g* $\wedge$ *more*) $\wedge$ *finite*); *f* $\wedge$
$\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *g* $\wedge$ *finite*))
by (*meson 36 40 Prop11 lift-imp-trans*)
have *50*: $\vdash$ *g* $\wedge$ *more* $\longrightarrow$ *more*
by *auto*
have *5*: $\vdash$ (*g* $\wedge$ *more*) $\wedge$ *finite* $\longrightarrow$ *more*
by (*simp add*: *50 Prop05 Prop07 finite-d-def*)
have *6*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ (*while* (*init w*) *do* *g* $\wedge$ *finite*)
using *4 5 ChopContraB* **by** *blast*
from *6* **show** *?thesis* **by** (*simp add*: *Prop12*)
qed

lemma *SWhileIntro*:
assumes $\vdash$ $\neg$ (*init w*) $\wedge$ *f* $\longrightarrow$ *empty*
$\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ (*g* $\wedge$ *more*)$\frown$ *f*
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *swhile* (*init w*) *do* *g*
proof $-$
have *1*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *while* (*init w*) *do* *g*
using *assms*
using *assms*
unfolding *schop-d-def*
using *WhileIntro*[*of w f g*]
by *blast*
have *2*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *while* (*init w*) *do* *g* $\wedge$ *finite*
using *1* **by** *auto*
show *?thesis*
using *2*
SWhile-While[*of LIFT* (*init w*) *g*]
While-import-finite[*of LIFT* (*init w*) *g*]
by *fastforce*
qed

lemma *WhileElim*:
assumes $\vdash$ $\neg$ (*init w*) $\wedge$ *empty* $\longrightarrow$ *g*
$\vdash$ *init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); *g* $\longrightarrow$ *g*
shows $\vdash$ *while* (*init w*) *do* *f* $\wedge$ *finite* $\longrightarrow$ *g*
proof $-$
have *1*: $\vdash$ (*while* (*init w*) *do* *f* $\wedge$ *finite*)=
((¬ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*)))

by (*rule WhileEqvEmptyOrChopWhile*)
hence *11*: $\vdash ((while\ (\ init\ w)\ \ do\ \ f \wedge finite) \wedge \neg\ g) =$
$(((\neg(init\ w) \wedge empty) \vee$
$(init\ w \wedge ((f \wedge more) \wedge finite);(while\ (init\ w)\ do\ f \wedge finite))) \wedge \neg\ g)$
by (*metis inteq-reflection lift-and-com*)
have *2*: $\vdash \neg\ (\ init\ w) \wedge\ \ empty\ \longrightarrow g$
using *assms* **by** *blast*
hence *21*: $\vdash \neg\ g \longrightarrow \neg(\neg\ (\ init\ w) \wedge\ \ empty)$
by *auto*
have *22*: $\vdash\ ((\neg\ (init\ w) \wedge empty) \vee$
$(init\ w \wedge ((f \wedge more) \wedge finite);(while\ (init\ w)\ do\ f \wedge finite))) \wedge \neg\ \ g \longrightarrow$
$(init\ w \wedge ((f \wedge\ \ more) \wedge finite\ );\ \ (while\ (\ init\ w)\ \ do\ \ f \wedge finite))$
using *21* **by** *auto*
have *23*: $\vdash (while\ (\ init\ w)\ \ do\ \ f \wedge finite) \wedge \neg\ g \longrightarrow$
$(init\ w \wedge ((f \wedge\ \ more) \wedge finite\ );\ \ (while\ (\ init\ w)\ \ do\ \ f \wedge finite)) \wedge \neg\ \ g$
using *11 21* **by** *fastforce*
have *3*: $\vdash (init\ w) \wedge (((f \wedge\ \ more) \wedge finite\ );\ g) \longrightarrow g$
using *assms* **by** *blast*
hence *31*: $\vdash \neg\ g \longrightarrow \neg((init\ w) \wedge (((f \wedge\ \ more) \wedge finite\ );\ g))$
by *fastforce*
have *32*: $\vdash (init\ w \wedge ((f \wedge\ \ more) \wedge finite\ );\ \ (while\ (\ init\ w)\ \ do\ \ f \wedge finite)) \wedge \neg\ g \longrightarrow$
$((((f \wedge\ \ more) \wedge finite\ );\ \ (while\ \ (init\ w)\ \ do\ \ f \wedge finite)) \wedge$
$\neg\ (((f \wedge\ \ more) \wedge finite\ );\ g)) \wedge \neg g$
using *31* **by** *fastforce*
have *4*: $\vdash\ \ (while\ \ (init\ w)\ \ do\ \ f \wedge finite) \wedge \neg\ \ g \longrightarrow$
$(((f \wedge\ \ more) \wedge finite\ );\ \ (while\ \ (init\ w)\ \ do\ \ f \wedge finite)) \wedge$
$\neg\ (((f \wedge\ \ more) \wedge finite\ );\ g)$
by (*meson 23 32 Prop12 lift-imp-trans*)
have *5*: $\vdash (f \wedge\ \ more) \wedge finite\ \longrightarrow\ \ more$
by *auto*
from *4 5* **show** *?thesis* **using**
ChopContraB[*of LIFT*(*while*\ \ (*init w*)\ \ *do*\ \ $f \wedge finite$) *LIFT*(*g*) *LIFT*(((*f* $\wedge$\ \ *more*) $\wedge$ *finite*))]
by *auto*
qed

lemma *SWhileElim*:
assumes $\vdash\ \neg\ (\ init\ w) \wedge\ \ empty\ \longrightarrow g$
$\vdash init\ w \wedge (f \wedge\ \ more\ ) \frown g \longrightarrow g$
shows $\vdash\ \ swhile\ \ (init\ w)\ do\ \ f \longrightarrow g$
using *assms*
unfolding *schop-d-def*
using *WhileElim*[*of w g f*] *SWhile-While*[*of LIFT* (*init w*) *f*]
While-import-finite[*of LIFT* (*init w*) *f*]
by *fastforce*

lemma *BaWhileImpWhile*:
$\vdash\ \ \ ba\ (f \longrightarrow g) \wedge finite \longrightarrow (\ while\ \ (init\ w)\ \ do\ \ f) \longrightarrow (\ while\ \ (init\ w)\ \ do\ \ g)$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow ((init\ w \wedge f)\ \longrightarrow (init\ w \wedge g))$

by *auto*
hence *2*: $\vdash$ *ba* $(f \longrightarrow g) \longrightarrow$ *ba* $((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
using *BaImpBa* **by** *blast*
have *3*: $\vdash$ *ba* $((init\ w \wedge f) \longrightarrow (init\ w \wedge g)) \wedge finite \longrightarrow ((init\ w \wedge f)^\star \longrightarrow (init\ w \wedge g)^\star)$
by (*rule BaCSImpCS*)
have *4*: $\vdash$ *ba* $(f \longrightarrow g) \wedge finite \longrightarrow ((init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ w))$
$\longrightarrow (init\ w \wedge g)^\star \wedge fin\ (\neg\ (init\ w)))$
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *SBaSWhileImpSWhile*:
$\vdash$ *sba* $(f \longrightarrow g) \longrightarrow (\ swhile\ \ (init\ w)\ \ do\ \ f) \longrightarrow (\ swhile\ \ (init\ w)\ \ do\ \ g)$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
by *auto*
hence *2*: $\vdash$ *sba* $(f \longrightarrow g) \longrightarrow$ *sba* $((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
by (*rule SBaImpSBa*)
have *3*: $\vdash$ *sba* $((init\ w \wedge f) \longrightarrow (init\ w \wedge g)) \longrightarrow$
$(schopstar\ (init\ w \wedge f) \longrightarrow schopstar\ (init\ w \wedge g))$
by (*rule SBaSCSImpSCS*)
have *4*: $\vdash$ *sba* $(f \longrightarrow g) \longrightarrow (schopstar\ (init\ w \wedge f) \wedge sfin\ (\neg\ (\ init\ w)) \longrightarrow$
$schopstar\ (init\ w \wedge g) \wedge sfin\ (\neg\ (init\ w)))$
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *swhile-d-def*)
qed

lemma *WhileImpWhile*:
assumes $\vdash$ $f \longrightarrow g$
shows $\vdash$ $(\ while\ \ (init\ w)\ \ do\ \ f) \wedge finite \longrightarrow (\ while\ \ (init\ w)\ \ do\ \ g)$
proof −
have *1*: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
hence *2*: $\vdash$ *ba* $(f \longrightarrow g)$
by (*rule BaGen*)
have *3*: $\vdash$ *ba* $(f \longrightarrow g) \wedge finite \longrightarrow (\ while\ \ (init\ w)\ \ do\ \ f) \longrightarrow (\ while\ \ (init\ w)\ \ do\ \ g)$
by (*rule BaWhileImpWhile*)
have *4*: $\vdash$ *ba* $(f \longrightarrow g) \longrightarrow (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite) \longrightarrow (\ while\ \ (init\ w)\ \ do\ \ g)$
using *3* **by** (*auto simp*: *Valid-def*)
from *2 4* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *SWhileImpSWhile*:
assumes $\vdash$ $f \longrightarrow g$
shows $\vdash$ $(\ swhile\ \ (init\ w)\ \ do\ \ f) \longrightarrow (\ swhile\ \ (init\ w)\ \ do\ \ g)$
proof −
have *1*: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
hence *2*: $\vdash$ *sba* $(f \longrightarrow g)$
by (*rule SBaGen*)

have *3*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*swhile* (*init w*) *do f*) $\longrightarrow$ (*swhile* (*init w*) *do g*)
by (*rule SBaSWhileImpSWhile*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *NotSkipEqvEmptyOrSkipChopMore*:
$\vdash$ ($\neg$ *skip*) = (*empty* $\vee$ *skip*;*more*)
using *ChopEmpty WnextEqvEmptyOrNext* **unfolding** *next-d-def wnext-d-def empty-d-def*
by (*metis inteq-reflection*)

lemma *FiniteInfSplitAnd*:
$\vdash$ *f* = ((*f* $\vee$ *inf*) $\wedge$ (*f* $\vee$ *finite*))
unfolding *finite-d-def* **by** *fastforce*

lemma *ChopAndInf-alt-1*:
$\vdash$ ((*f*;*g*) $\wedge$ *inf*) = (*f*;(*g* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *inf*);(*g* $\wedge$ *finite*))
proof −
have *1*: $\vdash$ ((*f*;*g*) $\wedge$ *inf*) = (((*f* $\wedge$ *finite*);*g* $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *inf*);*g* $\wedge$ *inf*))
by (*metis ChopAndInf OrChopEqv OrFiniteInf inteq-reflection*)
have *2*: $\vdash$ ((*f* $\wedge$ *finite*);*g* $\wedge$ *inf*) = (*f* $\wedge$ *finite*);(*g* $\wedge$ *inf*)
by (*simp add*: *ChopAndInf*)
have *3*: $\vdash$ ((*f* $\wedge$ *inf*);*g* $\wedge$ *inf*) = ((*f* $\wedge$ *inf*);(*g* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *inf*);(*g* $\wedge$ *finite*))
by (*metis AndInfChopEqvAndInf ChopAndInf int-simps*(*27*) *inteq-reflection*)
have *4*: $\vdash$ ((*f* $\wedge$ *finite*);(*g* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *inf*);(*g* $\wedge$ *inf*)) = (*f*;(*g* $\wedge$ *inf*))
by (*metis 1 ChopAndInf inteq-reflection*)
have *5*: $\vdash$ (((*f* $\wedge$ *finite*);*g* $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *inf*);*g* $\wedge$ *inf*)) =
(*f*;(*g* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *inf*);(*g* $\wedge$ *finite*))
using *2 3 4* **by** *fastforce*
show *?thesis*
by (*meson 1 5 Prop04 Prop11*)
qed

lemma *ChopAndInf-alt-2*:
$\vdash$ ((*f*;*g*) $\wedge$ *inf*) = ((*f* $\wedge$ *finite*);(*g* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *inf*);(*g* $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *inf*);(*g* $\wedge$ *finite*)))
proof −
have *1*: $\vdash$ (*f*;(*g* $\wedge$ *inf*)) = ((*f* $\wedge$ *finite*);(*g* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *inf*);(*g* $\wedge$ *inf*))
using *OrChopEqvRule OrFiniteInf* **by** *blast*
show *?thesis* **using** *ChopAndInf-alt-1 1* **by** *fastforce*
qed

lemma *AndEmptyChopAndMore*:
$\vdash$ ((*f* $\wedge$ *empty*);*g* $\wedge$ *more*) = (*f* $\wedge$ *empty*);(*g* $\wedge$ *more*)
proof −
have *1*: $\vdash$ (*f* $\wedge$ *empty*);*g* = (*init f* $\wedge$ *g*)
by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop*)
have *2*: $\vdash$ ((*f* $\wedge$ *empty*);*g* $\wedge$ *more*) = (*init f* $\wedge$ *g* $\wedge$ *more*)
using *1* **by** *fastforce*
have *3*: $\vdash$ (*init f* $\wedge$ *g* $\wedge$ *more*) = (*f* $\wedge$ *empty*);(*g* $\wedge$ *more*)
by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop*)
show *?thesis* **using** *2 3*

by (*meson Prop04 Prop11*)
qed

lemma *OrChopOr*:
$\vdash$ (*f1* $\vee$ *f2*);(*g1* $\vee$ *g2*) = (*f1*;*g1* $\vee$ *f1*;*g2* $\vee$ *f2*;*g1* $\vee$ *f2*;*g2*)
using *ChopOrEqv*[*of LIFT* (*f1* $\vee$ *f2*) *g1 g2*] *OrChopEqv*[*of f1 f2 g1*] *OrChopEqv*[*of f1 f2 g2*]
by *fastforce*

lemma *ChopAndMore-alt-1*:
$\vdash$ (*f*;*g* $\wedge$ *more*) = ((*f* $\wedge$ *empty*);(*g* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*);*g*)
proof −
have *1*: $\vdash$ (*f*;*g*) = ((*f* $\wedge$ *empty*);*g* $\vee$ (*f* $\wedge$ *more*);*g*)
using *EmptyOrMoreSplit OrChopEqvRule* **by** *blast*
have *2*: $\vdash$ (*f*;*g* $\wedge$ *more*) = (((*f* $\wedge$ *empty*);*g* $\wedge$ *more*) $\vee$ ((*f* $\wedge$ *more*);*g* $\wedge$ *more*))
using *1* **by** *fastforce*
have *3*: $\vdash$ ((*f* $\wedge$ *empty*);*g* $\wedge$ *more*) = (*f* $\wedge$ *empty*);(*g* $\wedge$ *more*)
using *AndEmptyChopAndMore* **by** *blast*
have *4*: $\vdash$ ((*f* $\wedge$ *more*);*g* $\wedge$ *more*) = ((*f* $\wedge$ *more*);*g*)
by (*meson AndChopB MoreChopImpMore Prop04 Prop10 int-simps*(*4*) *lift-imp-trans*)
show *?thesis*
by (*metis 2 3 4 int-eq*)
qed

lemma *ChopAndMore-alt-2*:
$\vdash$ (*f*;*g* $\wedge$ *more*) = ((*f* $\wedge$ *empty*);(*g* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*);(*g* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*);(*g* $\wedge$ *more*))
using *ChopAndMore-alt-1*[*of f g*]
by (*meson ChopOrEqvRule EmptyOrMoreSplit Prop06*)

lemma *WPowerstar-And-Inf-Eqv-WPowerplus-And-Inf*:
$\vdash$ (*wpowerstar f* $\wedge$ *inf*) = (*wpowerstar f*;*f* $\wedge$ *inf*)
proof −
have *1*: $\vdash$ *wpowerstar f* = (*empty* $\vee$ *wpowerstar f*;*f*)
by (*meson ChopEmpty Prop04 WPowerstar-unfoldr-eq*)
have *2*: $\vdash$ ((*empty* $\vee$ *wpowerstar f*;*f*) $\wedge$ *inf*) = (*wpowerstar f*;*f* $\wedge$ *inf*)
using *EmptyImpFinite* **unfolding** *finite-d-def* **by** *fastforce*
show *?thesis* **using** *1 2*
by *fastforce*
qed

lemma *WPowerstar-Inf-Import*:
$\vdash$ ((*wpowerstar f*) $\wedge$ *inf*) = ((*wpowerstar f*) $\wedge$ *finite*);(*f* $\wedge$ *inf*)
proof −
have *2*: $\vdash$ ((*istar f*) $\wedge$ *inf*) = ((*wpowerstar f*) $\wedge$ *inf*)
by (*metis IStarWPowerstar int-eq lift-and-com*)
have *3*:$\vdash$ (*wpowerstar f*) = (*empty* $\vee$ (*wpowerstar f*);*f*)
by (*meson WPowerstar-unfoldR WPowerstar-unfoldr-eq int-iffD2 int-iffI*)
have *4*: $\vdash$ ((*empty* $\vee$ (*wpowerstar f*);*f*) $\wedge$ *inf*) = ((*wpowerstar f*);*f* $\wedge$ *inf*)
using *EmptyImpFinite* **unfolding** *finite-d-def* **by** *fastforce*

have *6*: $\vdash$ *empty* $\longrightarrow$ *inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
using *EmptyImpFinite* **unfolding** *finite-d-def* **by** *fastforce*
have *61*: $\vdash$ (*f*;((*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\wedge$ *inf*) =
((*f* $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *finite*);(((*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\wedge$ *inf*)))
by (*simp add*: *ChopAndInfB*)
have *62*: $\vdash$ ((*f* $\wedge$ *finite*);(((*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\wedge$ *inf*)) =
((*f* $\wedge$ *finite*);(((*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*))))
by (*metis* (*no-types*, *lifting*) *ChopAndInfB Prop03 Prop10 Prop12 inteq-reflection*)
have *63*: $\vdash$ (*f* $\wedge$ *inf*) $\wedge$ *inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*meson ChopEqvChop Prop03 Prop10 Prop11 Prop12 WPowerstarChopEqvChopOrRule WPowerstar-import-finite lift-and-com*)
have *64*: $\vdash$ ((*f* $\wedge$ *finite*);(((*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))) $\wedge$ *inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*metis* (*no-types*, *lifting*) *ChopAndInf ChopAssoc EmptyOrChopEqv Prop03 Prop12 WPowerstarEqv WPowerstar-import-finite inteq-reflection*)
have *7*: $\vdash$ *f*;((*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\wedge$ *inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
using *61 63 64 ChopAndInf* **by** *fastforce*
have *8*: $\vdash$ (*f*;*finite* $\wedge$ *inf*) = (*f* $\wedge$ *inf*)
using *ChopAndInfB*[*of f LIFT finite*] **unfolding** *finite-d-def*
by (*metis AndInfEqvChopFalse ChopAndInf FiniteInfSplitAnd finite-d-def int-simps*(*26*) *inteq-reflection lift-and-com*)
have *9*: $\vdash$ *f*;*finite* $\wedge$ *inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*metis 7 8 ChopAndInf-alt-1 PowerstarEqvSemhelp2 Prop03 inteq-reflection lift-imp-trans*)
have *10*: $\vdash$ (*inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) =
(*finite* $\vee$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*))
unfolding *finite-d-def* **by** *fastforce*
have *11*: $\vdash$ *f*;(*inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) = ((*f*;*finite*) $\vee$ *f*;((*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))
by (*metis 10 ChopOrEqv inteq-reflection*)
have *12*: $\vdash$ *empty* $\vee$ *f*;(*inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\longrightarrow$ *inf* $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*metis 11 6 7 9 Prop02 Prop09 int-eq*)
have *13*:$\vdash$ ((*istar f*)) $\longrightarrow$ (*inf* $\longrightarrow$ ((*wpowerstar f*) $\wedge$ *finite*);(*f* $\wedge$ *inf*))
using *IStarWeakInduct*[*of f LIFT* (*inf* $\longrightarrow$ ((*wpowerstar f*) $\wedge$ *finite*);(*f* $\wedge$ *inf*))]
using *12* **by** *blast*
have *14*:$\vdash$ ((*wpowerstar f*) $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\longrightarrow$ ((*wpowerstar f*);*f* $\wedge$ *inf*)
by (*meson AndChopA ChopAndInf int-iffD2 lift-imp-trans*)
show *?thesis*
by (*metis 10 13 14 2 3 4 InfEqvNotFinite Prop07 int-iffI inteq-reflection*)
qed

lemma *Prop-Var-Empty-Finite-Import*:
$\vdash$ ((\$*p* $\wedge$ *empty*) $\wedge$ *finite*) = (\$*p* $\wedge$ *empty*)
using *EmptyImpFinite* **by** *auto*

lemma *Not-Prop-Var-And-Empty-Inf*:
$\vdash$ $\neg$((\$*p* $\wedge$ *empty*) $\wedge$ *inf*)
using *FiniteAndEmptyEqvEmpty InfEqvNotFinite* **by** *fastforce*

lemma *Init-And-Finite-Absorb*:

$\vdash$ *init* (*f* $\wedge$ *finite*) = *init f*
by (*metis EmptySChop Initprop*(*1*) *init-d-def int-simps*(*17*) *inteq-reflection lift-and-com schop-d-def*)

lemma *Chop-And-More*:
$\vdash$ (*f*;*g* $\wedge$ *more*) = ((*f* $\wedge$ *more*);*g* $\vee$ *f*;(*g* $\wedge$ *more*))
proof −
have *1*: $\vdash$ *f*;*g* =
((*f* $\wedge$ *empty*);(*g* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *empty*);(*g* $\wedge$ *more*) $\vee$
(*f* $\wedge$ *more*);(*g* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*);(*g* $\wedge$ *more*))
by (*meson ChopEqvChop EmptyOrMoreSplit OrChopOr Prop04 Prop11*)
have *2*: $\vdash$ (*f*;*g* $\wedge$ *more*) = (
((*f* $\wedge$ *empty*);(*g* $\wedge$ *empty*) $\wedge$ *more*) $\vee$
((*f* $\wedge$ *empty*);(*g* $\wedge$ *more*) $\wedge$ *more*) $\vee$
((*f* $\wedge$ *more*);(*g* $\wedge$ *empty*) $\wedge$ *more*) $\vee$
((*f* $\wedge$ *more*);(*g* $\wedge$ *more*) $\wedge$ *more*))
using *1* **by** *fastforce*
have *3*: $\vdash$ (((*f* $\wedge$ *empty*);(*g* $\wedge$ *empty*) $\wedge$ *more*)) =
((*f*;*g* $\wedge$ *empty*) $\wedge$ *more*)
by (*metis ChopAndEmptyEqvEmptyChopEmpty int-simps*(*1*) *inteq-reflection*)
have *4*: $\vdash$ ¬ ((*f*;*g* $\wedge$ *empty*) $\wedge$ *more*)
unfolding *empty-d-def* **by** *fastforce*
have *5*:$\vdash$ ((*f* $\wedge$ *empty*);(*g* $\wedge$ *more*) $\wedge$ *more*) = ((*f* $\wedge$ *empty*);(*g* $\wedge$ *more*))
by (*meson ChopAndB ChopEmpty ChopMoreImpMore Prop04 Prop10 lift-imp-trans*)
have *6*: $\vdash$ ((*f* $\wedge$ *more*);(*g* $\wedge$ *empty*) $\wedge$ *more*) = ((*f* $\wedge$ *more*);(*g* $\wedge$ *empty*))
using *AndChopB MoreChopImpMore* **by** *fastforce*
have *7*: $\vdash$ ((*f* $\wedge$ *more*);(*g* $\wedge$ *more*) $\wedge$ *more*) = ((*f* $\wedge$ *more*);(*g* $\wedge$ *more*))
by (*meson ChopAndB ChopMoreImpMore Prop10 Prop11 lift-imp-trans*)
have *8*: $\vdash$ (*f* $\wedge$ *more*);*g* = ((*f* $\wedge$ *more*);(*g* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*);(*g* $\wedge$ *more*))
using *ChopOrEqvRule EmptyOrMoreSplit* **by** *blast*
have *9*: $\vdash$ *f*;(*g* $\wedge$ *more*) = ((*f* $\wedge$ *empty*);(*g* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*);(*g* $\wedge$ *more*))
by (*simp add*: *EmptyOrMoreSplit OrChopEqvRule*)
have *10*: $\vdash$ ((*f* $\wedge$ *more*);*g* $\vee$ *f*;(*g* $\wedge$ *more*)) =
((*f* $\wedge$ *more*);(*g* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*);(*g* $\wedge$ *more*) $\vee$
(*f* $\wedge$ *empty*);(*g* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*);(*g* $\wedge$ *more*))
using *8 9* **by** *fastforce*
have *11*: $\vdash$ (((*f* $\wedge$ *empty*);(*g* $\wedge$ *more*)) $\vee$ ((*f* $\wedge$ *more*);(*g* $\wedge$ *empty*)) $\vee$ ((*f* $\wedge$ *more*);(*g* $\wedge$ *more*))) =
((*f* $\wedge$ *more*);*g* $\vee$ *f*;(*g* $\wedge$ *more*))
using *10 8 9* **by** *fastforce*
show *?thesis*
by (*meson 11 ChopAndMore-alt-2 Prop11 lift-imp-trans*)
qed

end

theory *Omega*
imports *Chopstar*
begin

This theory defines the omega operator for infinite ITL and provides a library of lemmas. We also

define a weak version womega that corresponds to the omega operator from Omega Algebra [1]. We also provide a semantic version aomega and provide lemmas that express various relationships between them. We also ported the numerous omega algebra lemmas from [1] to ITL.

3.6 Omega and variants

3.6.1 Definitions

Weak Omega

lemma *MoreChopSemMono* [*mono*]:
$(w \models (f \wedge more)\ ;g) =$
$(\ (\exists n.\ 0<n \wedge enat\ n \leq nlength\ w \wedge f\ (\ (ntaken\ n\ w))\ \wedge g\ \ (ndropn\ n\ \ w)) \vee$
$(\neg nfinite\ w \wedge f\ w))$
by (*simp add*: *itl-defs*)
(*metis bot-nat-0.not-eq-extremum le-zero-eq nlength-eq-enat-nfiniteD the-enat.simps zero-enat-def*)

coinductive *womega-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
for *F* **where**
$(s \models (F \wedge more);(womega\text{-}d\ F)) \Longrightarrow (s \models (womega\text{-}d\ F))$

syntax
-womega-d :: *lift* ⇒ *lift* ((-$^{\mathcal{W}}$) [*85*] *85*)

syntax (*ASCII*)
-womega-d :: *lift* ⇒ *lift* ((*womega* -) [*85*] *85*)

translations
-womega-d ⇌ *CONST womega-d*

lemma *WOmegaIntros*:
$\vdash (f \wedge more);(womega\ f) \longrightarrow (womega\ f)$
using *womega-d.intros* **using** *unl-lift2* **by** *blast*

lemma *WOmegaCases*:
$(w \models (womega\ F)) \Longrightarrow$
$(w \models (F \wedge more);(womega\ F) \Longrightarrow P) \Longrightarrow P$
using *womega-d.cases*[*of F w P*] **by** *auto*

lemma *WOmegaUnrollSem*:
$(s \models (womega\ f)) = (s \models (f \wedge more);(womega\ f))$
using *womega-d.cases*[*of f*]
by (*metis womega-d.simps*)

lemma *WOmegaUnroll*:
$\vdash (womega\ f) = (f \wedge more);(womega\ f)$
using *WOmegaUnrollSem*

using *unl-lift2* **by** *blast*

lemma *WOmegaCoinductSem*:
assumes $\bigwedge x.\ \ X\ x \Longrightarrow x \models\ (F \wedge more);(X\ \vee womega\ F)$
shows $\quad x \models X\ \longrightarrow womega\ F$
using *assms womega-d.coinduct*[*of X x F*]
by (*auto simp add*: *chop-defs*)

lemma *WOmegaCoinduct*:
assumes $\vdash X \longrightarrow\ (F \wedge more);(X \vee (womega\ F))$
shows $\ \vdash X\ \longrightarrow (womega\ F)$
using *assms WOmegaCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

lemma *WOmegaWeakCoinductSem*:
assumes $\bigwedge x.\ \ x \models X \Longrightarrow x \models\ (F \wedge more);X$
shows $\ x \models X\ \longrightarrow womega\ F$
using *assms womega-d.coinduct*[*of X x F*]
by (*auto simp add*: *chop-defs*)

lemma *WOmegaWeakCoinduct*:
assumes $\vdash X \longrightarrow\ (F \wedge more);X$
shows $\ \vdash X\ \longrightarrow (womega\ F)$
using *assms WOmegaWeakCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

Omega

lemma *FMoreSem-var* [*mono*]:
$(w \models\ ((f \wedge more) \wedge finite);g) =$
$((0< nlength\ w\ \wedge (\exists n.\ f\ (\ (ntaken\ n\ w)) \wedge 0<n \wedge g\ \ (ndropn\ n\ \ w)))\ )$
by (*simp add*: *itl-defs*)
(*metis enat-0-iff*(*1*) *linorder-not-less ndropn-all neq0-conv*
nfinite-nlength-enat nfinite-ntaken ntaken-all order-le-less)

definition *omega-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *omega-d F* ≡ *LIFT*(*womega*(*F* ∧ *finite*))

syntax
-omega-d :: *lift* ⇒ *lift* ((-$^{\omega}$) [*85*] *85*)

syntax (*ASCII*)
-omega-d :: *lift* ⇒ *lift* ((*omega* -) [*85*] *85*)

translations
-omega-d ⇌ *CONST omega-d*

lemma *Chop-more-finite-swap*:
$\vdash ((f \wedge more) \wedge finite) = ((f \wedge finite) \wedge more)$

by *fastforce*

lemma *OmegaIntros*:
$\vdash\ (f \wedge more)^\frown(omega\ f) \longrightarrow (omega\ f)$
unfolding *omega-d-def schop-d-def*
by (*meson Chop-more-finite-swap LeftChopImpChop Prop11 WOmegaIntros lift-imp-trans*)

lemma *OmegaCases*:
$(w \models (omega\ F)) \Longrightarrow$
$(w \models\ (F \wedge more)^\frown(omega\ F) \Longrightarrow P) \Longrightarrow P$
unfolding *omega-d-def schop-d-def* **using** *womega-d.cases*[*of LIFT* $(F \wedge finite)$ *w*]
by (*metis Chop-more-finite-swap inteq-reflection*)

lemma *OmegaUnrollSem*:
$(s \models (omega\ f)) = (s \models (f \wedge more)^\frown(omega\ f))$
unfolding *omega-d-def schop-d-def*
by (*metis Chop-more-finite-swap WOmegaUnroll inteq-reflection*)

lemma *OmegaCoinductSem*:
assumes $\bigwedge x.\ X\ x \Longrightarrow x \models\ (F \wedge more)^\frown(X\ \vee\ omega\ F)$
shows $x \models X \longrightarrow omega\ F$
using *assms* **unfolding** *omega-d-def schop-d-def*
by (*metis Chop-more-finite-swap WOmegaCoinductSem inteq-reflection*)

lemma *OmegaWeakCoinductSem*:
assumes $\bigwedge x.\ X\ x \Longrightarrow x \models\ (F \wedge more)^\frown X$
shows $x \models X \longrightarrow omega\ F$
using *assms* **unfolding** *omega-d-def schop-d-def*
by (*metis Chop-more-finite-swap WOmegaWeakCoinductSem inteq-reflection*)

lemma *OmegaUnroll*:
$\vdash f^\omega = (f \wedge more)^\frown f^\omega$
using *OmegaUnrollSem unl-lift2* **by** *blast*

lemma *OmegaCoinduct*:
assumes $\vdash X \longrightarrow\ (F \wedge more)^\frown(X \vee (omega\ F))$
shows $\vdash X \longrightarrow (omega\ F)$
using *assms OmegaCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

lemma *OmegaWeakCoinduct*:
assumes $\vdash X \longrightarrow\ (F \wedge more)^\frown X$
shows $\vdash X \longrightarrow (omega\ F)$
using *assms OmegaWeakCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

Alternative definition for Omega

lemma *infinite-nidx-imp-infinite-interval*:
assumes $\neg$ *nfinite l*

nidx l
(nnth l 0) = 0
∀ i. (nnth l i) ≤ nlength s
shows *¬ nfinite s*
proof
assume *nfinite s*
thus *False*
using *assms*
proof (*induct s rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis dual-order.antisym enat-ord-simps(1) linorder-linear nfinite-ntaken nidx-gr-first nlength-NNil not-gr-zero ntaken-all zero-le zero-less-Suc*)
next
case (*NCons x nell*)
then show *?case*
proof −
have *1*: *⋀j. (nnth l j) < (nnth l (Suc j))*
by (*metis assms(1) assms(2) linorder-le-cases nfinite-ntaken nidx-expand ntaken-all*)
have *2*: *∀ i. enat (nnth l i) ≤ nlength (NCons x nell) ⟹ False*
by (*metis 1 NCons.hyps(2) assms(1) assms(2) assms(3) dual-order.strict-iff-order enat-ord-simps(1) iless-Suc-eq linorder-not-le nlength-NCons*)
show *?thesis*
using *2 NCons.prems(4)* **by** *auto*
qed
qed
qed

definition *aomega-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *aomega-d F* ≡
(*λs.*
(∃ (*l*:: *nat nellist*).
¬nfinite l ∧ (nnth l 0) = 0 ∧ nidx l ∧
(∀ i. (nnth l i) ≤ nlength s) ∧
(∀ i. ((nsubn s (nnth l i) (nnth l (Suc i))) ⊨ F))
)
)

syntax
-aomega-d :: *lift* ⇒ *lift* ((*aomega -*) [*85*] *85*)

syntax (*ASCII*)
-aomega-d :: *lift* ⇒ *lift* ((*aomega -*) [*85*] *85*)

translations
-aomega-d ⇌ *CONST aomega-d*

lemma *aomega-unroll-chain-a*:

assumes $(\exists\, l.\ \neg\ nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))))$
shows $(\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (\ (ntaken\ n\ \sigma)) \wedge$
$0 < n \wedge$
$enat\ 0 < nlength\ \sigma \wedge$
$(\exists\, l.$
$\neg\ nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l\ \wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ n) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
$)$
$)$
proof −
obtain l **where** *1*: $\neg nfinite\ l \wedge\ (nnth\ l\ 0) = 0 \wedge nidx\ \ l \wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge (\forall\, i.\ f\ (\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
using *assms* **by** *auto*
have *2*: $l = NCons\ (nnth\ l\ 0)\ (ndropn\ 1\ l)$
by (*metis 1 One-nat-def gr-zeroI ndropn-0 ndropn-Suc-conv-ndropn nlength-eq-enat-nfiniteD zero-enat-def*)
have *3*: $(nnth\ l\ 0) = 0$
using *1* **by** *blast*
have *4*: $(nnth\ l\ 0) < (nnth\ l\ 1)$
using *1*
by (*metis One-nat-def Suc-ile-eq enat-defs(1) nidx-gr-first nlength-eq-enat-nfiniteD not-gr-zero zero-less-one*)
have *5*: $nidx\ (ndropn\ 1\ l)$
using *1* *nidx-expand*[*of l*] *nidx-expand*[*of* $(ndropn\ 1\ l)$]
by (*metis dual-order.order-iff-strict enat-defs(1) ndropn-all ndropn-nnth nfinite-ndropn-b nlength-NNil nlength-eq-enat-nfiniteD not-le-imp-less plus-1-eq-Suc*)
have *6*: $f\ (\ (nsubn\ \sigma\ (nnth\ l\ 0)\ (nnth\ l\ 1)))$
by (*metis 1 One-nat-def*)
have *7*: $(\forall\, i.\ f\ (\ (nsubn\ \sigma\ \ (nnth\ (ndropn\ 1\ l)\ i)\ (nnth\ (ndropn\ 1\ l)\ (Suc\ i)))))$
by (*simp add*: *1*)
have *8*: $f\ (\ (ntaken\ (nnth\ l\ 1)\ \sigma))$
by (*metis 1 4 One-nat-def Suc-diff-1 Suc-diff-Suc ndropn-0 nsubn-def1*)
have *81*: $\neg\ nfinite\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))$
by (*simp add*: *1*)
have *82*: $\bigwedge j.\ 0 < j \longrightarrow (nnth\ l\ 1) < nnth\ (ndropn\ 1\ l)\ j$
by (*metis 1 Suc-diff-1 enat-defs(1) linorder-le-cases ndropn-all ndropn-nnth nfinite-ndropn-b nidx-less nlength-NNil nlength-eq-enat-nfiniteD plus-1-eq-Suc*)
have *83*: $\bigwedge j.\ nnth\ (nmap\ (\lambda e.\ e - nnth\ l\ 1)\ (ndropn\ 1\ l))\ j =$
$(nnth\ l\ (Suc\ j)) - (nnth\ l\ 1)$
by (*metis 81 le-cases ndropn-nnth nfinite-ntaken nlength-nmap nnth-nmap ntaken-all plus-1-eq-Suc*)
have *84*: $\bigwedge j.\ nnth\ (nmap\ (\lambda e.\ e - nnth\ l\ 1)\ (ndropn\ 1\ l))\ (Suc\ j) =$
$(nnth\ l\ (Suc\ (Suc\ j))) - (nnth\ l\ 1)$
using *83* **by** *blast*
have *840*: $\bigwedge j.\ (nnth\ l\ 1) \leq (nnth\ l\ (Suc\ j))$
by (*metis 1 diff-add-zero diff-is-0-eq linorder-le-cases nfinite-ntaken nidx-less-eq ntaken-all plus-1-eq-Suc*)

have *85*: $\bigwedge j.\ (nnth\ l\ (Suc\ j)) - (nnth\ l\ 1) < (nnth\ l\ (Suc\ (Suc\ j))) - (nnth\ l\ 1)$
by (*metis 1 840 diff-less-mono linorder-le-cases nfinite-ntaken nidx-expand ntaken-all*)
have *86*: $(\forall\, i.\ enat\ (Suc\ i) \leq nlength\ (nmap\ (\lambda e.\ e - nnth\ l\ 1)\ (ndropn\ 1\ l)) \longrightarrow$
$nnth\ (nmap\ (\lambda e.\ e - nnth\ l\ 1)\ (ndropn\ 1\ l))\ i <$
$nnth\ (nmap\ (\lambda e.\ e - nnth\ l\ 1)\ (ndropn\ 1\ l))\ (Suc\ i))$
using *83 85* **by** *presburger*
have *9*: $nidx\ \ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))$
using *nidx-expand*[*of* $(nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))$]
using *83 85* **by** *presburger*
have *91*: $\bigwedge j.\ (nnth\ (nmap\ (\lambda e.\ e + (nnth\ l\ 1))\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l)))\ j) =$
$(nnth\ l\ (Suc\ j))$
by (*metis 81 82 83 One-nat-def add-Suc diff-Suc-1 diff-Suc-eq-diff-pred*
diff-add diff-is-0-eq le-refl linorder-le-cases ndropn-nnth nfinite-ntaken nlength-nmap
nnth-nmap not-le-imp-less ntaken-all order-less-imp-le plus-1-eq-Suc)
have *10*: $(\forall\, i.\ f\ (\ (nsubn\ \sigma$
$(nnth\ (nmap\ (\lambda e.\ e + (nnth\ l\ 1))\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l)))\ i)$
$(nnth\ (nmap\ (\lambda e.\ e + (nnth\ l\ 1))\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l)))\ (Suc\ i))\ )))$
using *1 91* **by** *presburger*
have *11*: $(\forall\, i.\ f\ (\ (nsubn\ \sigma$
$(\ (nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i)\ + (nnth\ l\ 1)\ )$
$(\ (nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ (Suc\ i))\ + (nnth\ l\ 1)\ )$
$)\ )\ )$
using *7 83 840* **by** *fastforce*
have *12*: $(\forall\, i.\ f\ (\ (nsubn\ (ndropn\ (nnth\ l\ 1)\ \sigma)$
$(\ (nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i)\ )$
$(\ (nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ (Suc\ i))\ )$
$)\ )\ )$
by (*metis 11 83 85 nsubn-ndropn*)
have *121*: $nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ 0 = 0$
using *83 One-nat-def* **by** *presburger*
have *122*: $\neg\ nfinite\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l)) \wedge$
$nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ 0 = 0 \wedge$
$nidx\ \ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l)) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ (ndropn\ (nnth\ l\ 1)\ \sigma)$
$(\ (nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i)\ )$
$(\ (nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ (Suc\ i))\ )$
$)\ )\ )$
using *12 121 81 9* **by** *blast*
have *123*: $(\forall\, i.\ (nnth\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i) \leq nlength\ (ndropn\ (nnth\ l\ 1)\ \sigma))$
by (*meson 1 enat-ile infinite-nidx-imp-infinite-interval le-cases nfinite-ndropn-b*
nlength-eq-enat-nfiniteD)
have *13*: $(\exists\, ls.$
$\neg\ nfinite\ ls \wedge (nnth\ ls\ 0) = 0 \wedge nidx\ ls\ \wedge$
$(\forall\, i.\ (nnth\ ls\ i) \leq nlength\ (ndropn\ (nnth\ l\ 1)\ \sigma)) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ (ndropn\ (nnth\ l\ 1)\ \sigma)\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))$
$)$
using *122 123* **by** *blast*
from *13* **show** *?thesis* **using** *4 8*
by (*metis 1 enat-ord-simps*(*1*) *linorder-not-less ndropn-nlength not-gr-zero zero-enat-def*)
qed

lemma *aomega-unroll-chain-b*:
assumes ($\exists\, n.$ *enat* $n \leq$ *nlength* $\sigma \wedge$
f ((*ntaken n* σ)) $\wedge$
$0 < n \wedge$
enat $0 <$ *nlength* $\sigma \wedge$
($\exists\, l.$
$\neg$*nfinite l* $\wedge$ (*nnth l* 0) $= 0 \wedge$ *nidx l* $\wedge$
($\forall\, i.$ (*nnth l i*) $\leq$ *nlength* $\sigma -$ *enat n*) $\wedge$
($\forall\, i.$ *f* ((*nsubn* (*ndropn n* σ) (*nnth l i*) (*nnth l* (*Suc i*)))))
)
)
shows ($\exists\, l.$ $\neg$ *nfinite l* $\wedge$ (*nnth l* 0) $= 0 \wedge$ *nidx l* $\wedge$
($\forall\, i.$ (*nnth l i*) $\leq$ *nlength* σ) $\wedge$
($\forall\, i.$ *f* ((*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))))))
proof −
obtain *n* **where** *1*: *enat* $n \leq$ *nlength* $\sigma \wedge$ *f* ((*ntaken n* σ)) $\wedge$
$0 < n \wedge$ *enat* $0 <$ *nlength* $\sigma \wedge$
($\exists\, l.$
$\neg$*nfinite l* $\wedge$ (*nnth l* 0) $= 0 \wedge$ *nidx l* $\wedge$
($\forall\, i.$ (*nnth l i*) $\leq$ *nlength* $\sigma -$ *enat n*) $\wedge$
($\forall\, i.$ *f* ((*nsubn* (*ndropn n* σ) (*nnth l i*) (*nnth l* (*Suc i*)))))
)
using *assms* **by** *auto*
have *2*: ($\exists\, l.$
$\neg$*nfinite l* $\wedge$ (*nnth l* 0) $= 0 \wedge$ *nidx l* $\wedge$
($\forall\, i.$ (*nnth l i*) $\leq$ *nlength* (*ndropn n* σ)) $\wedge$
($\forall\, i.$ *f* ((*nsubn* (*ndropn n* σ) (*nnth l i*) (*nnth l* (*Suc i*)))))
)
using *1* **by** *auto*
obtain *l* **where** *3*: $\neg$*nfinite l* $\wedge$ (*nnth l* 0) $= 0 \wedge$ *nidx l* $\wedge$
($\forall\, i.$ (*nnth l i*) $\leq$ *nlength* (*ndropn n* σ)) $\wedge$
($\forall\, i.$ *f* ((*nsubn* (*ndropn n* σ) (*nnth l i*) (*nnth l* (*Suc i*)))))
using *2* **by** *auto*
have *4*: *nidx* (*nmap* ($\lambda e.$ $e + n$) *l*)
using *3*
by (*simp add*: *Suc-ile-eq nidx-expand*)
have *42*: $\bigwedge j.$ *nnth* (*NCons 0* (*nmap* ($\lambda e.$ $e + n$) *l*)) *j* =
(*case j of* $0 \Rightarrow 0$ |
(*Suc k*) $\Rightarrow$ *nnth* (*nmap* ($\lambda e.$ $e + n$) *l*) *k*)
by (*simp add*: *nnth-NCons*)
have *43*: $\bigwedge j.$ *nnth* (*NCons 0* (*nmap* ($\lambda e.$ $e + n$) *l*)) (*Suc j*) =
nnth (*nmap* ($\lambda e.$ $e + n$) *l*) *j*
by *simp*
have *44*: $0 <$ *nnth* (*nmap* ($\lambda e.$ $e + n$) *l*) 0
using *1 enat-defs*(*1*) **by** *auto*
have *45*: $\bigwedge j.$ *nnth* (*nmap* ($\lambda e.$ $e + n$) *l*) *j* $<$ *nnth* (*nmap* ($\lambda e.$ $e + n$) *l*) (*Suc j*)
by (*metis 3 4 add.right-neutral le-cases nfinite-ntaken nidx-less nlength-nmap ntaken-all*)
have *46*: ($\forall\, i.$ *enat* (*Suc i*) $\leq$ *nlength* (*NCons 0* (*nmap* ($\lambda e.$ $e + n$) *l*)) $\longrightarrow$

nnth (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) *i* <
nnth (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) (*Suc i*))
by (*metis 43 44 45 lessI less-Suc-eq-0-disj nnth-0*)
have *5*: *nidx* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*))
using *46 nidx-expand* **by** *blast*
have *6*: ((*ntaken n σ*)) =
((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) *0*)
(*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) (*Suc 0*))))
by (*metis 3 43 44 One-nat-def Suc-diff-1 Suc-diff-Suc add-0 ndropn-0 nnth-0 nnth-nmap nsubn-def1 zero-enat-def zero-le*)
have *7*: (∀ *i*. *f* ((*nsubn* (*ndropn n σ*) (*nnth l i*) (*nnth l* (*Suc i*)))))
using *3* **by** *auto*
have *8*: (∀ *i*. *f* ((*nsubn σ* ((*nnth l i*)+*n*) ((*nnth l* (*Suc i*)) +*n*))))
using *7*
by (*simp add*: *add.commute ndropn-ndropn nsubn-def1*)
have *9*: (∀ *i*. *f* ((*nsubn σ* ((*nnth* (*nmap* (λ*e*. *e* + *n*) *l*) *i*))
((*nnth* (*nmap* (λ*e*. *e* + *n*) *l*) (*Suc i*))))))
using *8*
by (*metis 45 Suc-ile-eq dual-order.order-iff-strict linorder-le-cases nat-neq-iff nlength-nmap nnth-beyond nnth-nmap*)
have *10*: *f* ((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) *0*)
(*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) (*Suc 0*))))
using *1 6* **by** *auto*
have *11*: (∀ *i*. *i*>*0* ⟶
f ((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) *i*)
(*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) (*Suc i*)))))
by (*metis 9 Suc-diff-1 nnth-Suc-NCons*)
have *12*: (∀ *i*.
f ((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) *i*)
(*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) (*Suc i*)))))
using *10 11 neq0-conv* **by** *blast*
have *13*: ∀ *i*. (*nnth* (*NCons 0* (*nmap* (λ*e*. *e* + *n*) *l*)) *i*) ≤ *nlength σ*
by (*metis 3 add.commute enat-ile infinite-nidx-imp-infinite-interval linorder-le-cases ndropn-all ndropn-ndropn nfinite-ndropn-b nlength-NNil nlength-eq-enat-nfiniteD zero-le*)
show *?thesis* **using** *12 5*
by (*metis 13 3 nfinite-NCons nfinite-nmap nnth-0*)
qed

lemma *aomega-unroll-chain*:
(∃ *l*. ¬ *nfinite l* ∧ (*nnth l 0*) = *0* ∧ *nidx l* ∧
(∀ *i*. (*nnth l i*) ≤ *nlength σ*) ∧
(∀ *i*. *f* ((*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))))
=
(∃ *n*. *enat n* ≤ *nlength σ* ∧
f ((*ntaken n σ*)) ∧
0 < *n* ∧
enat 0 < *nlength σ* ∧
(∃ *l*.
¬ *nfinite l* ∧ (*nnth l 0*) = *0* ∧ *nidx l* ∧
(∀ *i*. (*nnth l i*) ≤ *nlength σ* − *enat n*) ∧

```
          (∀ i. f ( (nsubn (ndropn n σ) (nnth l i) (nnth l (Suc i)))))
       )
  )

using aomega-unroll-chain-a[of σ f]  aomega-unroll-chain-b[of σ f]
by blast

lemma aomega-unroll-sem:
 (σ ⊨ ((f ∧ more) ∧ finite);(aomega f) = (aomega f))
proof
 (simp add: itl-defs zero-enat-def aomega-d-def)
 show ((∃ n. enat n ≤ nlength σ ∧
         f (ntaken n σ) ∧
         0 < n ∧
         enat 0 < nlength σ ∧
         (∃ l. ¬ nfinite l ∧ nnth l 0 = 0 ∧ nidx l ∧
             (∀ i. enat (nnth l i) ≤ nlength σ − enat n) ∧
             (∀ i. f (nsubn (ndropn n σ) (nnth l i) (nnth l (Suc i)))))) ∨
    ¬ nfinite σ ∧ f σ ∧ enat 0 < nlength σ ∧ nfinite σ) =
   (∃ l. ¬ nfinite l ∧ nnth l 0 = 0 ∧ nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ) ∧
        (∀ i. f (nsubn σ (nnth l i) (nnth l (Suc i)))))
  using aomega-unroll-chain[of σ f] by blast
qed

lemma AOmegaUnroll:
⊢ (aomega f) = ((f ∧ more) ∧ finite);(aomega f)
unfolding Valid-def
using aomega-unroll-sem by auto

lemma AOmegaInductSem-help:
 (σ ⊨  inf ∧ g ∧ □(g ⟶ ((f ∧ more) ∧ finite);g)) =
  ( ¬ nfinite σ ∧ g σ ∧
             (∀ n. g ( (ndropn n σ)) ⟶
                    (∃ na. f ( (nsubn σ n (na+n))) ∧
                        0 < na ∧ g ( (ndropn (n + na) σ))))
  )

by (simp add: itl-defs zero-enat-def min-def ndropn-ndropn nsubn-def1)
   (metis linorder-le-cases ndropn-all ndropn-nlength nfinite-NNil nfinite-ndropn-b )


primrec cpoint :: ('a::world) formula ⇒ 'a formula ⇒ nat ⇒ 'a intervals ⇒ nat
 where cpoint f g 0 σ = 0
     | cpoint f g (Suc n) σ =
       (ϵ x. (∃ m. ( (nsubn σ (cpoint f g n σ) (m+(cpoint f g n σ))  ) ⊨ f)
             ∧ m>0 ∧((ndropn (m+(cpoint f g n σ)) σ) ⊨ g) ∧
             x=m+(cpoint f g n σ)
            )
       )
```

lemma *cpoint-expand-0*:
(*cpoint f g 0 σ*) =*0*
by *simp*

lemma *cpoint-expand-1*:
(*cpoint f g 1 σ*) =
(*SOME x.* (∃ *m. f* ((*nsubn σ 0* (*m*)))
∧ *m*>*0* ∧ *g* ((*ndropn* (*m*) *σ*))
∧ *x*=*m*))
by (*simp add*: *itl-defs*)

lemma *cpoint-expand-n*:
(*cpoint f g* (*Suc n*) *σ*) =
(*SOME x.* (∃ *m. f* ((*nsubn σ* (*cpoint f g n σ*) (*m*+(*cpoint f g n σ*))))
∧ *m*>*0* ∧ *g* ((*ndropn* (*m*+(*cpoint f g n σ*)) *σ*))
∧ *x*=*m*+(*cpoint f g n σ*))
)
by (*simp add*: *itl-defs*)

lemma *cpoint-0*:
assumes ¬*nfinite σ* ∧ *g σ* ∧
(∀ *k. g* ((*ndropn k σ*)) ⟶
(∃ *m. f* ((*nsubn σ k* (*m*+*k*))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+*k*) *σ*))))

shows *g* ((*ndropn* (*cpoint f g i σ*) *σ*))
proof
(*induct i*)
case *0*
then show *?case* **by** (*simp add*: *assms*)
next
case (*Suc i*)
then show *?case*
proof −
have *1*: *g* ((*ndropn* (*cpoint f g i σ*) *σ*))
by (*simp add*: *Suc.hyps*)
have *2*: *g* ((*ndropn* (*cpoint f g i σ*) *σ*)) ⟶
(∃ *m. f* ((*nsubn σ* (*cpoint f g i σ*) (*m*+(*cpoint f g i σ*)))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+(*cpoint f g i σ*)) *σ*)))
using *assms* **by** *blast*
have *3*: (∃ *m. f* ((*nsubn σ* (*cpoint f g i σ*) (*m*+(*cpoint f g i σ*)))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+(*cpoint f g i σ*)) *σ*)))
using *1 2* **by** *auto*
have *4*: (*cpoint f g* (*Suc i*) *σ*) =
(*SOME x.* (∃ *m. f* ((*nsubn σ* (*cpoint f g i σ*) (*m*+(*cpoint f g i σ*))))
∧ *m*>*0* ∧ *g* ((*ndropn* (*m*+(*cpoint f g i σ*)) *σ*))

```
                ∧ x=m+(cpoint f g i σ)))
 by simp
 have 5: g ((ndropn ((cpoint f g (Suc i) σ)) σ))
 using 3 4 someI-ex[of λx. (∃ m. f ( (nsubn σ (cpoint f g i σ) (m+(cpoint f g i σ))))
                ∧ m>0 ∧ g ((ndropn (m+(cpoint f g i σ)) σ))
                ∧ x=m+(cpoint f g i σ))] by auto
 from 5 show ?thesis by auto
 qed
qed



lemma cpoint-1:
 assumes ¬ nfinite σ ∧ g  σ ∧
              (∀ k. g ( (ndropn k σ)) ⟶
                    (∃ m. f ( (nsubn σ k (m+k))) ∧
                          0 < m ∧ g ( (ndropn (m+k) σ))))

 shows   ( g ((ndropn (cpoint f g i σ) σ))
           ⟹ g ((ndropn (cpoint f g (Suc i) σ) σ)))
using assms cpoint-0 by blast

lemma cpoint-2:
 assumes ¬nfinite σ ∧ g  σ ∧
              (∀ k. g ( (ndropn k σ)) ⟶
                    (∃ m. f ( (nsubn σ k (m+k))) ∧
                          0 < m ∧ g ( (ndropn (m+k) σ))))

 shows   f ( (nsubn σ (cpoint f g i σ) (cpoint f g (Suc i) σ)))

proof
 (induct i)
 case 0
 then show ?case
 proof −
  have 1: g ((ndropn 0 σ))
      using assms cpoint-0 cpoint-expand-0 by force
  have 2: (∃ m. f ( (nsubn σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)))) ∧
                          0 < m ∧ g ( (ndropn (m+(cpoint f g 0 σ)) σ)))
      using assms 1 by (metis cpoint-expand-0)
  have 3: (cpoint f g 1 σ) =
         (SOME x. (∃ m. f ((nsubn σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)) ))
                    ∧ m>0 ∧ g ( (ndropn (m+(cpoint f g 0 σ)) σ))
                    ∧ x=m+(cpoint f g 0 σ))
               )

      by simp
  have 4: f ( (nsubn σ (cpoint f g 0 σ) ((cpoint f g 1 σ)) ))
      using 2 3 someI-ex[of λx. (∃ m. f ( (nsubn σ (cpoint f g 0 σ)
                                                (m+(cpoint f g 0 σ)) ))
```

```
                    ∧ m>0 ∧ g ( (ndropn (m+(cpoint f g 0 σ)) σ))
                    ∧ x=m+(cpoint f g 0 σ))] by auto
 from 4 show ?thesis by auto
 qed
next
case (Suc i)
then show ?case
proof −
 have n1: g ( (ndropn (cpoint f g (Suc i) σ) σ))
     using assms cpoint-0 by blast
 have n2: (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ)))) ∧
                    0 < m ∧ g ( (ndropn (m+(cpoint f g (Suc i) σ)) σ)))
     using assms n1 by auto
 have n3: (cpoint f g (Suc (Suc i)) σ) =
       (SOME x. (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
            ∧ m>0 ∧ g ((ndropn (m+(cpoint f g (Suc i) σ)) σ))
            ∧ x=m+(cpoint f g (Suc i) σ))
          )
     using cpoint-expand-n by blast
 have n4: f ( (nsubn σ (cpoint f g (Suc i) σ) ((cpoint f g (Suc (Suc i)) σ))))
     using n2 n3 someI-ex[of λx. (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ)
                                                (m+(cpoint f g (Suc i) σ))))
            ∧ m>0 ∧ g ((ndropn (m+(cpoint f g (Suc i) σ)) σ))
            ∧ x=m+(cpoint f g (Suc i) σ)) ] by auto
 from n4 show ?thesis by auto
 qed
qed

lemma cpoint-3a:
 m>0 ∧ x=m+(cpoint f g (Suc i) σ) ⟹ (cpoint f g (Suc i) σ) < x
by auto


lemma cpoint-3:
 assumes ¬nfinite σ ∧ g  σ ∧
           (∀ k. g ( (ndropn k σ)) ⟶
                 (∃ m. f ( (nsubn σ k (m+k))) ∧
                     0 < m ∧ g ( (ndropn (m+k) σ))))

 shows  (cpoint f g i σ) < (cpoint f g (Suc i) σ)

proof
 (induct i)
 case 0
 then show ?case
 proof −
  have 1: g ((ndropn 0 σ))
      using assms cpoint-0 cpoint-expand-0 by force
  have 2: (∃ m. f ( (nsubn σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)))) ∧
                    0 < m ∧ g ( (ndropn (m+(cpoint f g 0 σ)) σ)))
```

```
    using assms 1 by (metis cpoint-expand-0)
  have 3: (cpoint f g 1 σ) =
        (SOME x. (∃ m. f ( (nsubn σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)) ))
                    ∧ m>0 ∧ g ( (ndropn (m+(cpoint f g 0 σ)) σ))
                    ∧ x=m+(cpoint f g 0 σ))
            )

    by simp
  have 4: (cpoint f g 0 σ) < (cpoint f g 1 σ)
    using 2 3 someI-ex[of λx. (∃ m. f ( (nsubn σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)) ))
                    ∧ m>0 ∧ g ( (ndropn (m+(cpoint f g 0 σ)) σ))
                    ∧ x=m+(cpoint f g 0 σ))] by auto
  from 4 show ?thesis by auto
  qed
 next
 case (Suc i)
 then show ?case
 proof −
  have n1: g ( (ndropn (cpoint f g (Suc i) σ) σ))
    using assms cpoint-0 by blast
  have n2: (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ)))) ∧
                    0 < m ∧ g ( (ndropn (m+(cpoint f g (Suc i) σ)) σ)))
    using assms n1 by auto
  have n3: (cpoint f g (Suc (Suc i)) σ) =
        (SOME x. (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
            ∧ m>0 ∧ g ((ndropn (m+(cpoint f g (Suc i) σ)) σ))
            ∧ x=m+(cpoint f g (Suc i) σ))
          )
    using cpoint-expand-n by blast
  have n4: (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
            ∧ m>0 ∧ g ((ndropn (m+(cpoint f g (Suc i) σ)) σ))
            ∧ (cpoint f g (Suc (Suc i)) σ)=m+(cpoint f g (Suc i) σ))
    using n2 n3 someI-ex[of λx. (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
            ∧ m>0 ∧ g ((ndropn (m+(cpoint f g (Suc i) σ)) σ))
            ∧ x=m+(cpoint f g (Suc i) σ)) ] by auto
  have n5: (cpoint f g (Suc i) σ) < (cpoint f g (Suc (Suc i)) σ)
  using n4 using cpoint-3a by blast
  from n5 show ?thesis by auto
  qed
qed


primcorec cpl :: ('a::world) formula ⇒ 'a formula ⇒ nat ⇒ 'a intervals ⇒ nat nellist
 where cpl f g x σ = NCons (cpoint f g x σ ) (cpl f g (Suc x) σ)

lemma
 nnth (cpl f g 0 σ) 0 = 0
by (metis cpl.disc-iff cpl.simps(2) cpoint-expand-0 nhd-conv-nnth)

lemma
```

nnth (*cpl f g 0* σ) *1* = *cpoint f g 1* σ
by (*metis One-nat-def cpl.code cpl.disc-iff cpl.simps*(*2*) *nhd-conv-nnth nnth-Suc-NCons*)

lemma *nnth-cpl*:
nnth (*cpl f g x* σ) *i* = *cpoint f g* (*x*+*i*) σ
proof (*induct i arbitrary*: *x*)
case *0*
then show *?case* **by** (*simp add*: *nnth-0-conv-nhd*)
next
case (*Suc i*)
then show *?case* **by** (*metis add-Suc-shift cpl.simps*(*3*) *nnth-ntl*)
qed

lemma *cpl-infinite*:
¬*nfinite* (*cpl f g x* σ)
proof
assume *nfinite* (*cpl f g x* σ)
thus *False*
proof (*induct zs*≡(*cpl f g x* σ) *arbitrary*: *x rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*metis cpl.disc-iff nellist.disc*(*1*))
next
case (*NCons x nell*)
then show *?case* **by** (*metis cpl.simps*(*3*) *nellist.sel*(*5*))
qed
qed

lemma *AOmegaInductSem*:
(*w* ⊨ (*inf* ∧ *g* ∧ □(*g* ⟶ ((*f* ∧ *more*) ∧ *finite*);*g*)) ⟶ *aomega f*)
proof −
have *1*: (*w* ⊨ (*inf* ∧ *g* ∧ □(*g* ⟶ ((*f* ∧ *more*) ∧ *finite*);*g*))) =
(¬*nfinite w* ∧ *g w* ∧
(∀ *k*. *g* ((*ndropn k w*)) ⟶
(∃ *m*. *f* ((*nsubn w k* (*m*+*k*))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+*k*) *w*)))))
using *AOmegaInductSem-help*[*of g f w*]
by (*simp add*: *add.commute*)
have *2*: (¬ *nfinite w* ∧ *g w* ∧
(∀ *k*. *g* ((*ndropn k w*)) ⟶
(∃ *m*. *f* ((*nsubn w k* (*m*+*k*))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+*k*) *w*)))))) ⟶
nidx (*cpl f g 0 w*)
using *nidx-expand*[*of* (*cpl f g 0 w*)] *nnth-cpl*[*of f g 0 w*]
by (*metis add-0 cpoint-3*)
have *3*: (¬*nfinite w* ∧ *g w* ∧
(∀ *k*. *g* ((*ndropn k w*)) ⟶
(∃ *m*. *f* ((*nsubn w k* (*m*+*k*))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+*k*) *w*)))))) ⟶

$$(\forall\, i.\ f\ (\ (nsubn\ w\ (nnth\ (cpl\ f\ g\ 0\ w)\ i)\ (nnth\ (cpl\ f\ g\ 0\ w)\ \ (Suc\ i)))))$$

using *1 cpoint-2* **by** (*metis add-0 nnth-cpl*)

have *31*: $(\neg nfinite\ w \wedge g\ w \wedge (\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow (\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge 0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow (\forall\, i.\ (nnth\ (cpl\ f\ g\ 0\ w)\ i) \leq nlength\ w)$

by (*simp add*: *nfinite-conv-nlength-enat*)

have *4*: $(\neg\ nfinite\ w \wedge g\ w \wedge (\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow (\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge 0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow \neg nfinite\ w \wedge \neg nfinite\ (cpl\ f\ g\ 0\ w) \wedge nnth\ (cpl\ f\ g\ 0\ w)\ 0 = 0 \wedge nidx\ \ (cpl\ f\ g\ 0\ w)\ \ \wedge (\forall\, i.\ (nnth\ (cpl\ f\ g\ 0\ w)\ i) \leq nlength\ w) \wedge (\forall\, i.\ f\ (\ (nsubn\ w\ (nnth\ (cpl\ f\ g\ 0\ w)\ \ i)\ (nnth\ (cpl\ f\ g\ 0\ w)\ \ (Suc\ i)))))$

using *2 3 31*

by (*simp add*: *cpl-infinite nnth-0-conv-nhd*)

have *5*: $(\neg nfinite\ w \wedge g\ w \wedge (\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow (\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge 0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow \neg nfinite\ w \wedge (\exists\, (ls).\ \ \neg nfinite\ ls \wedge nnth\ ls\ 0 = 0 \wedge\ \ nidx\ ls \wedge (\forall\, i.\ (nnth\ ls\ i) \leq nlength\ w) \wedge (\forall\, i.\ f\ (\ (nsubn\ w\ (nnth\ ls\ \ i)\ (nnth\ ls\ \ (Suc\ i))))) \wedge ls = (cpl\ f\ g\ 0\ w))$

using *4* **by** *auto*

have *6*: $(\neg nfinite\ w \wedge g\ w \wedge (\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow (\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge 0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow (w \models aomega\ f)$

using *3 5* **unfolding** *aomega-d-def* **by** *blast*

have *7*: $(w \models (inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)) \longrightarrow (aomega\ f)\ \ )$

using *6 1* **by** *auto*

from *7* **show** *?thesis* **by** *blast*

qed

lemma *AOmegaInduct*:

$\vdash (inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)) \longrightarrow aomega\ f$

unfolding *Valid-def* **using** *AOmegaInductSem*[*of g f*] **by** *blast*

lemma *AOmegaWeakCoInduct*:

assumes $\vdash g \longrightarrow ((f \wedge more) \wedge finite);g$

shows $\vdash inf \wedge g \longrightarrow aomega\ f$

proof −

have *1*: $\vdash \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)$
using *assms* **by** (*simp add*: *BoxGen*)
show *?thesis* **using** *assms AOmegaInduct*[*of g f*]
using *1* **by** *fastforce*
qed

lemma *Omega-WOmega*:
$\vdash (omega\ f) = (womega\ (f \wedge finite))$
unfolding *omega-d-def* **by** *simp*

lemma *WOmegaInducthelp*:
$\vdash (\ g \wedge \Box(g \longrightarrow f;g))$
$\longrightarrow$
$f;(\ (\ g \wedge \Box(g \longrightarrow f;g))\ )$
proof −
have *1*: $\vdash (\ g \wedge \Box(g \longrightarrow f;g)) = (\ g \wedge (g \longrightarrow f;g) \wedge \Box(g \longrightarrow f;g))$
using *BoxEqvAndBox*[*of LIFT g* $\longrightarrow$ *f*;*g*] **by** *fastforce*
have *2*: $\vdash (\ g \wedge (g \longrightarrow f;g) \wedge \Box(g \longrightarrow f;g)) = (\ g \wedge (\ f;g) \wedge \Box(g \longrightarrow f;g))$
by *fastforce*
have *4*: $\vdash \Box(g \longrightarrow f;g) = \Box\ (\Box\ (g \longrightarrow f;g))$
by (*simp add*: *BoxEqvBoxBox*)
have *5*: $\vdash (\ g \wedge (\ f;g) \wedge \Box(g \longrightarrow f;g)) = (\ g \wedge (\ f;g) \wedge\Box(\ \Box(g \longrightarrow f;g)))$
using *4* **by** *fastforce*
have *6*: $\vdash (\ g \wedge (\ f;g) \wedge \Box(\Box(g \longrightarrow f;g))) \longrightarrow f;(\ (\ g \wedge \Box(g \longrightarrow f;g))\ )$
using *ChopAndBoxImport*[*of f g LIFT* $\Box(g \longrightarrow f;g)$]
by (*meson Prop01 Prop05*)
show *?thesis*
by (*metis 1 2 5 6 int-eq*)
qed

lemma *OmegaInducthelp*:
$\vdash (\ g \wedge \Box(g \longrightarrow ((f \wedge more))^\frown g))$
$\longrightarrow$
$((f \wedge more))^\frown(\ (\ g \wedge \Box(g \longrightarrow ((f \wedge more))^\frown g))\ )$
using *WOmegaInducthelp* **unfolding** *schop-d-def* **by** *blast*

lemma *WOmegaInduct*:
$\vdash (\ g \wedge \Box(g \longrightarrow (f \wedge more);g)) \longrightarrow womega\ f$
by (*simp add*: *WOmegaInducthelp WOmegaWeakCoinduct*)

lemma *OmegaInduct*:
$\vdash (\ g \wedge \Box(g \longrightarrow ((f \wedge more))^\frown g)) \longrightarrow omega\ f$
by (*simp add*: *OmegaInducthelp OmegaWeakCoinduct*)

lemma *WOmega-coinduct*:
assumes $\vdash g \longrightarrow h \vee (f \wedge more);g$
shows $\vdash g \longrightarrow (womega\ f) \vee (wpowerstar\ (f \wedge more));h$
proof −
have *1*: $\vdash (wpowerstar\ (f \wedge more));h = (empty \vee (f \wedge more);(wpowerstar\ (f \wedge more)));h$
by (*simp add*: *LeftChopEqvChop WPowerstarEqv*)
have *2*: $\vdash (empty \vee (f \wedge more);(wpowerstar\ (f \wedge more)));h = (h \vee ((f \wedge more);(wpowerstar\ (f \wedge more)));h)$

by (*simp add*: *EmptyOrChopEqv*)
have *3*: $\vdash (\neg(h \vee ((f \wedge more);(wpowerstar\ (f \wedge more)));h)) = (\neg\ h \wedge \neg((f \wedge more);((wpowerstar\ (f \wedge more));h)))$
using *ChopAssoc* **by** *fastforce*
have *31*: $\vdash (\neg(\ (wpowerstar\ (f \wedge more));h)) = (\neg\ h \wedge \neg((f \wedge more);((wpowerstar\ (f \wedge more));h)))$
by (*metis 1 2 3 int-eq*)
have *32*: $\vdash ((\neg\ h \wedge \neg((f \wedge more);((wpowerstar\ (f \wedge more));h))) \wedge (h \vee (f \wedge more);g)) \longrightarrow$
$(\neg((f \wedge more);((wpowerstar\ (f \wedge more));h)) \wedge (f \wedge more);g)$
by *force*
have *33*: $\vdash\ g \wedge \neg(\ (wpowerstar\ (f \wedge more));h) \longrightarrow ((\neg\ h \wedge \neg((f \wedge more);((wpowerstar\ (f \wedge more));h))) \wedge (h \vee (f \wedge more);g))$
using *assms 31* **by** *fastforce*
have *4*: $\vdash g \wedge \neg(\ (wpowerstar\ (f \wedge more));h) \longrightarrow \neg((f \wedge more);((wpowerstar\ (f \wedge more));h)) \wedge (f \wedge more);g$
using *assms 31 32 33*
using *lift-imp-trans* **by** *blast*
have *5*: $\vdash \neg((f \wedge more);((wpowerstar\ (f \wedge more));h)) \wedge (f \wedge more);g \longrightarrow (f \wedge more);(g \wedge \neg\ wpowerstar\ (f \wedge more);h)$
by (*metis ChopAndNotChopImp int-eq lift-and-com*)
have *6*: $\vdash g \wedge \neg(\ (wpowerstar\ (f \wedge more));h) \longrightarrow (f \wedge more);(g \wedge \neg\ wpowerstar\ (f \wedge more);h)$
using *4 5 lift-imp-trans* **by** *blast*
have *7*: $\vdash g \wedge \neg((wpowerstar\ (f \wedge more));h) \longrightarrow (womega\ f)$
using *WOmegaWeakCoinduct*[*of LIFT* $g \wedge \neg((wpowerstar\ (f \wedge more));h)\ f$] *6*
by *blast*
show *?thesis* **using** *7* **by** *fastforce*
qed

lemma *WOmega-coinduct-var*:
assumes $\vdash g \longrightarrow h \vee (f \wedge more);g$
shows $\vdash g \longrightarrow (womega\ f) \vee (wpowerstar\ f);h$
using *assms*
by (*metis WOmega-coinduct WPowerstar-more-absorb int-eq*)

lemma *WOmega-coinduct-no-more*:
assumes $\vdash g \longrightarrow f;g$
shows $\vdash g \longrightarrow (womega\ f) \vee (wpowerstar\ f);init\ f$
proof −
have *1*: $\vdash (\ f) = (\ (f \wedge empty) \vee (f \wedge more))$
using *EmptyOrMoreSplit* **by** *auto*
have *2*: $\vdash\ (\ f;g) = (\ (f \wedge empty);g \vee (f \wedge more);g)$
by (*simp add*: *1 OrChopEqvRule*)

have *3*: $\vdash ((f \wedge empty);g) = (g \wedge init\ f)$
by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop lift-and-com*)
have *4*: $\vdash g \longrightarrow init\ f \vee (f \wedge more);g$
using *2 3 assms* **by** *fastforce*
show *?thesis* **using** *4 WOmega-coinduct-var* **by** *blast*
qed

lemma *WOmega-coinduct-no-more-var*:
assumes $\vdash g \longrightarrow h \vee f;g$
shows $\vdash g \longrightarrow (womega\ f) \vee (wpowerstar\ f);(h \vee init\ f)$
proof −
have *1*: $\vdash (\ f) = (\ (f \wedge empty) \vee (f \wedge more))$
using *EmptyOrMoreSplit* **by** *auto*
have *2*: $\vdash\ (\ f;g) = (\ (f \wedge empty);g \vee (f \wedge more);g)$
by (*simp add*: *1 OrChopEqvRule*)
have *3*: $\vdash ((f \wedge empty);g) = (g \wedge init\ f)$
by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop lift-and-com*)
have *4*: $\vdash g \longrightarrow (h \vee init\ f) \vee (f \wedge more);g$
using *assms 2 3* **by** *fastforce*
show *?thesis* **using** *4 WOmega-coinduct-var* **by** *blast*
qed

lemma *Omega-coinduct*:
assumes $\vdash g \longrightarrow h \vee (f \wedge more)^\frown g$
shows $\vdash g \longrightarrow (omega\ f) \vee (schopstar\ f)^\frown h$
using *assms WOmega-coinduct*[*of g h LIFT* $(f \wedge finite)$]
Omega-WOmega[*of f*] *SChopstar-WPowerstar*[*of LIFT* $(f \wedge more)$]
by (*metis Chop-more-finite-swap Prop10 SChopstar-WPowerstar SChopstar-finite WPowerstar-more-absorb inteq-reflection schop-d-def*)

NFusecat definition of Omega

definition *bomega-d* :: $('a::world)\ formula \Rightarrow 'a\ formula$
where *bomega-d F* ≡
$(\lambda s.$
$(\exists\, ss.$
$\neg nfinite\ ss \wedge s = nfusecat\ ss \wedge$
$nlastnfirst\ ss \wedge$
$(\forall\, i.\ nfinite\ (nnth\ ss\ i) \wedge nlength\ (nnth\ ss\ i) > 0) \wedge$
$(\forall\, i.\ (\ (nnth\ ss\ i) \models F)\ )$
$)$
$)$

syntax
-bomega-d :: *lift* ⇒ *lift* ((*bomega -*) [*85*] *85*)

syntax (*ASCII*)
-bomega-d :: *lift* ⇒ *lift* ((*bomega -*) [*85*] *85*)

translations
-bomega-d $\rightleftharpoons$ *CONST bomega-d*

lemma *aomega-bomega-help1*:
assumes $\neg$ *nfinite ss*
σ = *nfusecat ss*
nlastnfirst ss
$\forall$ *i. nfinite* (*nnth ss i*) $\wedge$ *nlength* (*nnth ss i*) $\neq$ *0*
$\forall$ *i. f* (*nnth ss i*)
shows $\exists$ *l.* $\neg$ *nfinite l* $\wedge$ *nnth l 0 = 0* $\wedge$ *nidx l* $\wedge$ ($\forall$ *i. enat* (*nnth l i*) $\leq$ *nlength* (*nfusecat ss*)) $\wedge$ ($\forall$ *i. f* (*nsubn* (*nfusecat ss*) (*nnth l i*) (*nnth l* (*Suc i*))))
proof −
have *1*: $\neg$*nfinite* (*addzero* (*lsum ss 0*))
using *assms* **using** *nfinite-lsum-conv*
by (*metis addzero-def nfinite-NCons*)
have *2*: *nnth* (*addzero* (*lsum ss 0*)) *0 =0*
using *assms lsum-nnth*[*of 0 ss 0*]
by (*metis lsum-addzero-nfirst nfirst-eq-nnth-zero*)
have *3*: *nidx* (*addzero* (*lsum ss 0*))
using *assms* **by** (*metis i0-less in-nset-conv-nnth lsum-addzero-nidx*)
have *4*: ($\forall$ *i. enat* (*nnth* (*addzero* (*lsum ss 0*)) *i*) $\leq$ *nlength* (*nfusecat ss*))
using *assms*
by (*metis enat-ile i0-less in-nset-conv-nnth linorder-linear nfinite-conv-nlength-enat nfusecat-nfinite*)
have *5*: $\bigwedge$*i. nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i* = *nnth ss i*
proof −
fix *i*
show *nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i* = *nnth ss i*
proof −
have *50*: *nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i* =
nsubn (*nfusecat ss*) (*nnth* ((*addzero* (*lsum ss 0*))) *i*) (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*))
by (*metis 1 nfinite-ntaken nle-le nnth-nsubn-nellist ntaken-all*)
have *51*: *nsubn* (*nfusecat ss*) (*nnth* ((*addzero* (*lsum ss 0*))) *i*) (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*)) =
ntaken (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*))
unfolding *nsubn-def1* **by** *simp*
have *52*: (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) =
nnth (*NCons 0* (*lsum ss 0*)) (*Suc i*) − *nnth* (*NCons 0* (*lsum ss 0*)) *i*
using *lsum-addzero-nnth*[*of i ss*] *lsum-addzero-nnth*[*of Suc i ss*]
by (*metis addzero-def assms*(*1*) *lsum-nlength*
nlength-eq-enat-nfiniteD zero-enat-def)
have *53*: *nnth* (*NCons 0* (*lsum ss 0*)) (*Suc i*) − *nnth* (*NCons 0* (*lsum ss 0*)) *i* =
nnth (*lsum ss 0*) *i* − (*if i = 0 then 0 else nnth* (*lsum ss 0*) (*i*−*1*))
using *assms* **apply** *simp*
by (*metis Suc-pred nnth-Suc-NCons*)
have *54*: *nnth* (*lsum ss 0*) *i* = ($\sum$ *k*::*nat*= *0..i. the-enat* (*nlength*(*nnth ss k*))))
using *assms*
by (*metis add-0 enat-ile in-nset-conv-nnth lsum-nnth*
nfinite-conv-nlength-enat nle-le)
have *55*: *i*>*0* $\Longrightarrow$ *nnth* (*lsum ss 0*) (*i*−*1*) = ($\sum$ *k*::*nat*= *0..*(*i*−*1*). (*the-enat* (*nlength*(*nnth ss k*))))
using *assms*

by (*metis add-0 enat-ile in-nset-conv-nnth lsum-nnth nfinite-conv-nlength-enat nle-le*)
have *56*: *i>0* $\Longrightarrow$
($\sum$ *k*::*nat*= *0..i*. (*the-enat* (*nlength*(*nnth ss k*)))) −
($\sum$ *k*::*nat*= *0..*(*i*−*1*). (*the-enat* (*nlength*(*nnth ss k*)))) = (*the-enat*(*nlength*(*nnth ss i*)))
by (*metis Suc-diff-1 add-diff-cancel-left' sum.atLeast0-atMost-Suc*)
have *57*: *i=0* $\Longrightarrow$ *nnth* (*lsum ss 0*) *i* − (*if i* = *0 then 0 else nnth* (*lsum ss 0*) (*i*−*1*)) =
(*the-enat*(*nlength*(*nnth ss i*)))
by (*metis* (*full-types*) *add-0 diff-zero lsum-nfirst nfirst-eq-nnth-zero*)
have *58*: *nnth* (*lsum ss 0*) *i* − (*if i* = *0 then 0 else nnth* (*lsum ss 0*) (*i*−*1*)) =
(*the-enat*(*nlength*(*nnth ss i*)))
using *54 55 56* **by** *auto*
have *59*: (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) =
(*the-enat*(*nlength*(*nnth ss i*)))
using *52 53 58* **by** *presburger*
have *60*: *ntaken* (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*) − *nnth* (*addzero* (*lsum ss 0*)) *i*) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*)) =
ntaken (*the-enat*(*nlength*(*nnth ss i*))) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*))
by (*simp add*: *59*)
have *61*: (*nnth* (*addzero* (*lsum ss 0*)) *i*) =
(*the-enat* (*if i* =*0 then 0 else* ($\sum$ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*))))))
using *lsum-addzero-nnth*[*of i ss*] *2 55 assms* **apply** *simp*
by (*metis Suc-pred addzero-def lsum-nlength*
nlength-eq-enat-nfiniteD nnth-Suc-NCons
zero-enat-def)
have *62*: (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*)) =
(*ndropn* (*the-enat* (*if i* =*0 then 0 else* ($\sum$ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*)))))))
(*nfusecat ss*))
using *61* **by** *auto*
have *63*: (*ndropn* (*the-enat* (*if i* =*0 then 0 else* ($\sum$ *j* = *0* .. (*i*−*1*) . (*the-enat*(*nlength*(*nnth ss j*)))))))
(*nfusecat ss*)) =
nfusecat (*ndropn i ss*)
using *nfusecat-ndropn*[*of i ss*] *assms 55 2 61* **apply** *auto*
by (*metis* (*no-types*, *lifting*) *ext in-nset-conv-nnth linorder-not-less nfinite-ntaken ntaken-all order-le-less sum-the-enat*)
have *64*: *ntaken* (*the-enat*(*nlength*(*nnth ss i*))) (*ndropn* (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nfusecat ss*)) =
ntaken (*the-enat*(*nlength*(*nnth ss i*))) (*nfusecat* (*ndropn i ss*))
using *62 63* **by** *presburger*
have *65*: $\bigwedge$*n*. ($\sum$ *ia* = *0..n*. (*the-enat* (*nlength* (*nnth* (*ndropn i ss*) *ia*)))) =
($\sum$ *ia* = *0..n*. (*the-enat* (*nlength* (*nnth ss* (*i*+ *ia*)))))
using *ndropn-nnth* **by** *auto*
have *66*: *nlength* (*ntaken* (*the-enat*(*nlength*(*nnth ss i*))) (*nfusecat* (*ndropn i ss*))) = (*the-enat*(*nlength*(*nnth ss i*)))
by (*metis 4 59 61 63 enat-minus-mono1 idiff-enat-enat min.absorb1 ndropn-nlength ntaken-nlength*)
have *67*: $\bigwedge$*j*. *enat j* ≤ (*the-enat*(*nlength*(*nnth ss i*))) $\Longrightarrow$
nnth (*ntaken* (*the-enat*(*nlength*(*nnth ss i*))) (*nfusecat* (*ndropn i ss*))) *j* =
nnth (*nfusecat* (*ndropn i ss*)) *j*
using *ntaken-nnth*
by (*metis enat-ord-simps*(*1*) *min.orderE*)
have *68*: $\bigwedge$*j*. *enat j* ≤ (*the-enat*(*nlength*(*nnth ss i*))) $\Longrightarrow$
nnth (*nfusecat* (*ndropn i ss*)) *j* =

nnth (*ndropn* (*the-enat*(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1)* . (*the-enat*(*nlength*(*nnth ss i*)))))) (*nfusecat ss*)) *j*
using *63* **by** *force*
have *69*: $\bigwedge$*j. enat j* ≤ (*the-enat*(*nlength*(*nnth ss i*))) ⟹
nnth (*ndropn* (*the-enat*(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1)* . (*the-enat*(*nlength*(*nnth ss i*)))))) (*nfusecat ss*)) *j* =
nnth (*nfusecat ss*) ((*the-enat*(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1)* . (*the-enat*(*nlength*(*nnth ss i*)))))))+*j*)
using *ndropn-nnth* **by** *simp*
have *70*: $\bigwedge$*j. enat j* ≤ (*the-enat*(*nlength*(*nnth ss i*))) ⟹
nnth (*nfusecat ss*) ((*the-enat*(*if i = 0 then 0 else* ($\sum$ *i = 0 .. (i−1)* . (*the-enat*(*nlength*(*nnth ss i*)))))))+*j*) =
nnth (*nfusecat ss*) (*nnth* (*addzero* (*lsum ss 0*)) *i* + *j*)
by (*simp add*: *61*)
have *71*: *nlastnfirst ss* ∧ *all-gr-zero ss* ∧ *all-nfinite ss*
by (*metis assms*(*3*,*4*) *i0-less in-nset-conv-nnth*)
have *72*: $\bigwedge$*j. enat j* ≤ *nlength* (*nnth ss i*) ⟹ *nnth* (*nfusecat ss*) (*nnth* (*addzero* (*lsum ss 0*)) *i* + *j*) = *nnth* (*nnth ss i*) *j*
using *71*
by (*metis assms*(*1*) *lsum-nfusecat-nnth-lsum-nnth nfinite-ntaken nle-le ntaken-all*)
have *73*: *nlength* (*nnth ss i*) =*nlength* (*nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i*)
by (*metis 50 60 64 66 assms*(*4*) *enat-ord-simps*(*3*)
enat-the-enat infinity-ileE nfinite-nlength-enat
nsubn-def1)
show *?thesis* **using** *72 73 nellist-eq-nnth-eq*[*of nnth* (*nsubn-nellist* (*nfusecat ss*) (*addzero* (*lsum ss 0*))) *i nnth ss i*]
by (*metis 50 51 59 61 63 66 67 69*)
qed
qed
have *5*: (∀ *i*. *f* (*nsubn* (*nfusecat ss*) (*nnth* (*addzero* (*lsum ss 0*)) *i*) (*nnth* (*addzero* (*lsum ss 0*)) (*Suc i*))))
using *assms*
by (*metis 1 5 nfinite-ntaken nle-le nnth-nsubn-nellist ntaken-all*)
show *?thesis*
using *1 2 3 4 5* **by** *auto*
qed

lemma *aomega-bomega-help2*:
assumes ¬ *nfinite l*
nnth l 0 = 0
nidx l
∀ *i*. *enat* (*nnth l i*) ≤ *nlength* *σ*
∀ *i*. *f* (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)))
shows∃ *ss*. ¬ *nfinite ss* ∧ *σ* = *nfusecat ss* ∧ *nlastnfirst ss* ∧ (∀ *i*. *nfinite* (*nnth ss i*) ∧ *nlength* (*nnth ss i*) ≠ *0*) ∧ (∀ *i*. *f* (*nnth ss i*))
proof −
have *1*: ¬ *nfinite* (*nfusecat* (*nsubn-nellist σ l*))
using *assms* **by** (*metis infinite-nidx-imp-infinite-interval ndropn-0 ndropn-nfirst nfusecat-nsubn-nellist-var*)

have *2*: *σ* = *nfusecat* (*nsubn-nellist σ l*)
using *assms*

by (*metis infinite-nidx-imp-infinite-interval nfirst-eq-nnth-zero nfusecat-nsubn-nellist-var*)
have *3*: *nlastnfirst* (*nsubn-nellist* σ *l*)
using *assms 1 2 nlastnfirst-nsubn-nellist-infinite* **by** *metis*
have *4*: ($\forall$ *i*. *nfinite* (*nnth* (*nsubn-nellist* σ *l*) *i*) $\wedge$ *nlength* (*nnth* (*nsubn-nellist* σ *l*) *i*) $\neq$ *0*)
using *assms*
by (*metis enat-ile linorder-linear nfinite-conv-nlength-enat nfinite-nnth-nsubn-nellist nidx-expand nnth-nsubn-nellis*
not-gr-zero nsubn-nlength-gr-one
zero-enat-def)
have *5*: ($\forall$ *i*. *f* (*nnth* (*nsubn-nellist* σ *l*) *i*))
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat nnth-nsubn-nellist*)
show *?thesis*
using *1 2 3 4 5 in-nset-conv-nnth nfusecat-nfinite not-gr-zero zero-enat-def* **by** *metis*
qed

lemma *aomega-bomega-sem*:
($\sigma \models$ (*aomega f*)) = ($\sigma \models$ (*bomega f*))
apply (*auto simp add*: *itl-defs bomega-d-def aomega-d-def*)
using *aomega-bomega-help2* **apply** *blast*
using *aomega-bomega-help1* **by** *blast*

lemma *aomega-eqv-bomega-sem*:
($\sigma \models$ (*aomega f*) = (*bomega f*))
using *aomega-bomega-sem* **by** *auto*

lemma *aomega-eqv-bomega*:
$\vdash$ (*aomega f*) = (*bomega f*)
using *aomega-eqv-bomega-sem* **unfolding** *Valid-def* **by** *auto*

3.6.2 Omega algebra

lemma *WOmega-coinduct-var1*:
assumes $\vdash f \longrightarrow$ *empty* $\vee$ (*g* $\wedge$ *more*);*f*
shows $\vdash f \longrightarrow$ (*womega g*) $\vee$ (*wpowerstar g*)
using *assms WOmega-coinduct-var*[*of f LIFT empty g*]
by (*metis ChopEmpty inteq-reflection*)

lemma *WOmega-coinduct-no-more-var1*:
assumes $\vdash f \longrightarrow$ *empty* $\vee$ *g*;*f*
shows $\vdash f \longrightarrow$ (*womega g*) $\vee$ (*wpowerstar g*);(*empty* $\vee$ *init g*)
using *assms WOmega-coinduct-no-more-var*[*of f LIFT empty g*] **by** *blast*

lemma *Omega-coinduct-var1*:
assumes $\vdash f \longrightarrow$ *empty* $\vee$ ((*g* $\wedge$ *more*)$\frown$*f*)
shows $\vdash f \longrightarrow$ (*omega g*) $\vee$ (*schopstar g*)
using *assms*
by (*metis Omega-coinduct Prop08 SChop-SChopstar-Closure SChopstar-imp-empty SChopstardef int-iffD1 schopstar-d-def*)

lemma *WOmega-coinduct-eq*:
assumes $\vdash f = (\ h \vee \quad (g \wedge more);f)$
shows $\vdash f \longrightarrow (womega\ g) \vee (wpowerstar\ g);h$
using *assms WOmega-coinduct-var int-iffD1* **by** *blast*

lemma *WOmega-coinduct-no-more-eq*:
assumes $\vdash f = (\ h \vee \quad g;f)$
shows $\vdash f \longrightarrow (womega\ g) \vee (wpowerstar\ g);(h \vee init\ g)$
using *assms* **by** (*simp add*: *Prop11 WOmega-coinduct-no-more-var*)

lemma *Omega-coinduct-eq*:
assumes $\vdash f = (\ h \vee \quad (g \wedge more)^\frown f)$
shows $\vdash f \longrightarrow (omega\ g) \vee (schopstar\ g)^\frown h$
using *assms* **by** (*simp add*: *Omega-coinduct Prop11*)

lemma *WOmega-coinduct-eq-var1*:
assumes $\vdash f = (\ empty \vee \quad (g \wedge more);f)$
shows $\vdash f \longrightarrow (womega\ g) \vee (wpowerstar\ g)$
using *assms WOmega-coinduct-var1 int-iffD1* **by** *blast*

lemma *WOmega-coinduct-no-more-eq-var1*:
assumes $\vdash f = (\ empty \vee \quad g;f)$
shows $\vdash f \longrightarrow (womega\ g) \vee (wpowerstar\ g);(empty \vee init\ g)$
using *assms WOmega-coinduct-no-more-eq* **by** *blast*

lemma *Omega-coinduct-eq-var1*:
assumes $\vdash f = (\ empty \vee \quad (g \wedge more)^\frown f)$
shows $\vdash f \longrightarrow (omega\ g) \vee (schopstar\ g)$
using *assms* **by** (*simp add*: *Omega-coinduct-var1 Prop11*)

lemma *WOmega-coinduct-eq-var2*:
assumes $\vdash f = (g \wedge more);f$
shows $\vdash f \longrightarrow (womega\ g)$
using *assms* **by** (*simp add*: *Prop11 WOmegaWeakCoinduct*)

lemma *WOmega-coinduct-no-more-eq-var2*:
assumes $\vdash f = g;f$
shows $\vdash f \longrightarrow (womega\ g) \vee (wpowerstar\ g);init\ g$
using *assms WOmega-coinduct-no-more int-iffD1* **by** *blast*

lemma *Omega-coinduct-eq-var2*:
assumes $\vdash f = (g \wedge more)^\frown f$
shows $\vdash f \longrightarrow (omega\ g)$
using *assms OmegaWeakCoinduct int-iffD1* **by** *blast*

lemma *WOmega-subdist*:
$\vdash (womega\ f) \longrightarrow (womega\ (f \vee g))$
proof −
have *0*: $\vdash ((f \vee g) \wedge more) = ((\ f \wedge more \vee g \wedge more))$
by *fastforce*

have *1*: $\vdash$ (*womega f*) $\longrightarrow$ ((*f* $\wedge$ *more* $\vee$ *g* $\wedge$ *more*));(*womega f*)
using *OrChopEqv WOmegaCases* **by** *fastforce*
have *15*: $\vdash$ ((*f* $\wedge$ *more* $\vee$ *g* $\wedge$ *more*));(*womega f*) = ((*f* $\vee$ *g*) $\wedge$ *more*);(*womega f*)
using *0* **by** (*metis LeftChopEqvChop int-eq*)
have *2*: $\vdash$ (*womega f*) $\longrightarrow$ ((*f* $\vee$ *g*) $\wedge$ *more*);(*womega f*)
by (*metis 0 1 int-eq*)
show *?thesis* **by** (*simp add*: *2 WOmegaWeakCoinduct*)
qed

lemma *Omega-subdist*:
$\vdash$ (*omega f*) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
by (*metis FiniteOr WOmega-subdist int-eq omega-d-def*)

lemma *WOmega-iso*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ (*womega f*) $\longrightarrow$ (*womega g*)
using *assms WOmega-subdist*[*of f g*]
by (*metis Prop02 Prop03 Prop11 int-simps*(*20*) *inteq-reflection*)

lemma *Omega-iso*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ (*omega f*) $\longrightarrow$ (*omega g*)
using *assms WOmega-iso*[*of LIFT* (*f* $\wedge$ *finite*) *LIFT* (*g* $\wedge$ *finite*)]
unfolding *omega-d-def*
by (*metis* (*no-types*, *opaque-lifting*) *AndChopA AndChopB ChopEmpty Prop12 inteq-reflection*
lift-imp-trans omega-d-def)

lemma *WOmega-subdist-var*:
$\vdash$ (*womega f*) $\vee$ (*womega g*) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*meson EmptyChop Prop02 Prop04 Prop05 Prop11 WOmega-iso WOmega-subdist*)

lemma *Omega-subdist-var*:
$\vdash$ (*omega f*) $\vee$ (*omega g*) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))

by (*meson Omega-iso Omega-subdist Prop02 Prop04 Prop05 int-iffD2 int-simps*(*26*))

lemma *WOmega-zero*:
$\vdash$ $\neg$(*womega #False*)
by (*metis AndInfChopEqvAndInf TrueW WOmegaUnroll int-simps*(*19*) *int-simps*(*3*) *inteq-reflection*)

lemma *Omega-zero*:
$\vdash$ $\neg$(*omega #False*)
using *Omega-WOmega*[*of LIFT #False*] *WOmega-zero* **by** *fastforce*

lemma *WOmega-star-1*:
$\vdash$ (*wpowerstar f*);(*womega f*) = (*womega f*)
proof −

have *1*: $\vdash (f \wedge more); (womega\ f) \longrightarrow (womega\ f)$
by (*simp add*: *WOmegaUnroll int-iffD2*)
have *2*: $\vdash (wpowerstar\ f);(womega\ f) \longrightarrow (womega\ f)$
by (*meson 1 LeftChopImpChop WPowerstar-inductL-var-equiv WPowerstar-more-absorb int-iffD2 lift-imp-trans*)
have *3*: $\vdash (womega\ f) \longrightarrow (wpowerstar\ f);(womega\ f)$
by (*metis 2 Prop03 Prop10 WPowerstarChopEqvChopOrRule inteq-reflection*)
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *Omega-star-1*:
$\vdash (schopstar\ f)^\frown(omega\ f) = (omega\ f)$
by (*meson OmegaUnroll Prop11 SChopstarMore-induct-lvar-eq2*)

lemma *WOmega-max-element*:
$\vdash f \longrightarrow (womega\ empty) \vee wpowerstar\ empty;init\ empty$
by (*metis EmptyChop WOmega-coinduct-no-more-eq-var2 inteq-reflection*)

lemma *WOmega-empty-zero*:
$\vdash \neg(womega\ empty)$
by (*metis Prop10 Prop12 WOmegaUnroll WOmegaWeakCoinduct WOmega-zero empty-d-def int-iffD2 int-simps*(*21*) *inteq-reflection lift-and-com*)

lemma *Omega-empty-zero*:
$\vdash \neg(omega\ empty)$
by (*metis FiniteAndEmptyEqvEmpty WOmega-empty-zero int-eq lift-and-com omega-d-def*)

lemma *WOmega-1*:
$\vdash (womega\ f);\ g \longrightarrow (womega\ f)$
by (*meson ChopAssoc LeftChopEqvChop Prop11 WOmegaUnroll WOmegaWeakCoinduct lift-imp-trans*)

lemma *Omega-1*:
$\vdash (omega\ f);\ g \longrightarrow (omega\ f)$
by (*simp add*: *WOmega-1 omega-d-def*)

lemma *WOmega-sup-id*:
assumes $\vdash empty \longrightarrow g$
shows $\vdash (womega\ f);g = (womega\ f)$
by (*meson ChopEmpty Prop11 RightChopImpChop WOmega-1 assms lift-imp-trans*)

lemma *Omega-sup-id*:
assumes $\vdash empty \longrightarrow g$
shows $\vdash (omega\ f);g = (omega\ f)$
using *assms*
by (*metis Omega-WOmega WOmega-sup-id int-eq*)

lemma *WOmega-simulation*:

assumes $\vdash h; (f \wedge more) \longrightarrow (g \wedge more)\ ;\ h$
shows $\vdash h\ ;\ (womega\ f) \longrightarrow (womega\ g)$
proof −
have *1*: $\vdash h\ ;\ (womega\ f) = h\ ;\ ((f \wedge more); (womega\ f))$
by (*simp add*: *RightChopEqvChop WOmegaUnroll*)
have *2*: $\vdash h\ ;\ ((f \wedge more); (womega\ f)) \longrightarrow (g \wedge more)\ ;\ (h; (womega\ f))$
by (*metis AndChopB ChopAssoc Prop10 assms inteq-reflection*)
have *3*: $\vdash h\ ;\ (womega\ f) \longrightarrow (g \wedge more)\ ;\ (h\ ;\ (womega\ f)\ )$
by (*metis 1 2 int-eq*)
show *?thesis* **by** (*simp add*: *3 WOmegaWeakCoinduct*)
qed

lemma *Omega-simulation*:
assumes $\vdash (h) \frown (f \wedge more) \longrightarrow (g \wedge more) \frown (h)$
shows $\vdash (h) \frown (omega\ f) \longrightarrow (omega\ g)$
using *assms*
by (*metis AndSChopA OmegaUnroll OmegaWeakCoinduct Prop10 SChopAssoc inteq-reflection lift-and-com*)

lemma *WOmega-WOmega*:
$\vdash (womega\ (womega\ f)) \longrightarrow (womega\ f)$
by (*meson AndChopA Prop11 WOmegaUnroll WOmega-1 lift-imp-trans*)

lemma *WOmega-eqv*:
assumes $\vdash f = g$
shows $\vdash womega\ f = womega\ g$
by (*meson Prop11 WOmega-iso assms*)

lemma *WOmega-More-absorb*:
$\vdash womega\ f = womega\ (f \wedge more)$
proof −
have *1*: $\vdash f^{\mathcal{W}} \longrightarrow (f \wedge more)^{\mathcal{W}}$
using *WOmega-coinduct-eq-var2*[*of LIFT womega f LIFT* $(f \wedge more)$]
by (*metis LeftChopImpChop Prop12 TrueW WOmegaUnroll WOmega-simulation int-simps*(*13*) *inteq-reflection*)
have *2*: $\vdash (f \wedge more)^{\mathcal{W}} \longrightarrow f^{\mathcal{W}}$
using *WOmega-coinduct-eq-var2*[*of LIFT womega* $(f \wedge more)$ *f*]
by (*metis AndChopB WOmegaUnroll WOmega-simulation inteq-reflection lift-and-com*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *WOmega-more-eqv*:
assumes $\vdash f = g$
shows $\vdash womega\ (f \wedge more) = womega\ g$
by (*meson Prop04 WOmega-More-absorb WOmega-eqv assms*)

lemma *WOmega-Wagner-1*:
$\vdash (womega\ (wpowerstar\ f)) = (womega\ f)$
proof −
have *1*: $\vdash (womega\ ((f \wedge more);wpowerstar\ f)) =$
$(((f \wedge more);(wpowerstar\ f));((f \wedge more);(wpowerstar\ f)));(womega\ ((f \wedge more);wpowerstar\ f))$
by (*metis* (*no-types*, *lifting*) *ChopAssoc WOmegaUnroll WOmega-star-1 WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)

have *2*: $\vdash (((f \wedge more);(wpowerstar\ f));((f \wedge more);(wpowerstar\ f)));(womega\ ((f \wedge more);wpowerstar\ f)) =$
$(f \wedge more);\ (womega\ ((f \wedge more);wpowerstar\ f))$
by (*metis* (*no-types*, *lifting*) *AndChopB ChopAssoc Prop11 WOmegaUnroll WPowerstar-inductL-var-equiv WPowerstar-invol WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop inteq-reflection lift-and-com*)

have *3*: $\vdash (womega\ ((f \wedge more);wpowerstar\ f)) \longrightarrow (womega\ f)$
by (*metis 1 2 WOmega-coinduct-eq-var2 inteq-reflection*)
have *4*: $\vdash (womega\ f) \longrightarrow (womega\ ((f \wedge more);wpowerstar\ f))$
by (*metis WOmega-More-absorb WOmega-iso WPowerstar-ext WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)
have *5*: $\vdash (wpowerstar\ f \wedge more) = (f \wedge more);powerstar\ f$
by (*metis ChopstarEqvPowerstar Chopstar-WPowerstar WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)
have *6*: $\vdash (womega\ (wpowerstar\ f)) = (womega\ ((f \wedge more);wpowerstar\ f))$
by (*metis 5 ChopstarEqvPowerstar Chopstar-WPowerstar WOmega-More-absorb WPowerstar-more-absorb inteq-reflection*)
show *?thesis* **using** *3 4 6* **by** *fastforce*
qed

lemma *Omega-Wagner-1*:
$\vdash (omega\ (schopstar\ f)) = (omega\ f)$
by (*metis Prop10 SChopstar-WPowerstar SChopstar-finite WOmega-Wagner-1 inteq-reflection omega-d-def*)

lemma *Omega-Eqv-rule*:
assumes $\vdash f = g$
shows $\vdash omega\ f = omega\ g$
using *assms*
by (*simp add*: *Omega-iso Prop11*)

lemma *Omega-Wagner-1-var2*:
$\vdash (omega\ ((f \wedge more)^\frown schopstar\ f)) = (omega\ f)$
proof −
have *1*: $\vdash ((f \wedge more)^\frown schopstar\ f \wedge finite \wedge more) = ((f \wedge more)^\frown schopstar\ f)$
by (*metis Prop10 Prop11 Prop12 SCSAndMoreEqvAndMoreSChop SChopImpFinite SChopstar-finite*)
have *2*: $\vdash ((schopstar\ f \wedge finite) \wedge more) = (f \wedge more)^\frown(schopstar\ f)$
by (*metis Prop10 SCSAndMoreEqvAndMoreSChop SChopstar-finite inteq-reflection*)
have *3*: $\vdash (omega\ ((f \wedge more)^\frown schopstar\ f)) = (omega\ (schopstar\ f))$
by (*metis 2 Omega-Wagner-1 SChopstar-and-more SChopstar-finite-absorb inteq-reflection*)
show *?thesis*

by (*metis 3 Omega-Wagner-1 inteq-reflection*)
qed

lemma *WOmega-Wagner-2-var*:
$\vdash (f \wedge more);(womega\ ((g \wedge more);(f \wedge more))) \longrightarrow (womega\ ((f \wedge more);(g \wedge more)))$
proof −
have *1*: $\vdash\ (f \wedge more);((g \wedge more);(f \wedge more) \wedge more) \longrightarrow ((f \wedge more);(g \wedge more) \wedge more);(f \wedge more)$
by (*metis* (*no-types*, *lifting*) *ChopAssoc Prop10 Prop12 RightChopImpMoreRule int-iffD1 inteq-reflection lift-and-com*)

show *?thesis* **using** *WOmega-simulation*[*of LIFT* $(f \wedge more)$ *LIFT* $((g \wedge more);(f \wedge more))$ *LIFT* $((f \wedge more);(g \wedge more))$]
using *1* **by** *blast*
qed

lemma *Omega-Wagner-2-var*:
$\vdash (f \wedge more)^\frown(omega\ ((g \wedge more)^\frown(f \wedge more))) \longrightarrow (omega\ ((f \wedge more)^\frown(g \wedge more)))$
using *WOmega-Wagner-2-var*[*of LIFT* $((f \wedge more) \wedge finite)$ *LIFT* $((g \wedge more) \wedge finite)$]
Omega-WOmega[*of LIFT* $(((g \wedge more) \wedge finite);((f \wedge more) \wedge finite))$]
Omega-WOmega[*of LIFT* $(((f \wedge more) \wedge finite);((g \wedge more) \wedge finite))$]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Omega-simulation SChopAssoc int-eq int-iffD1*)

lemma *WOmega-Wagner-2*:
$\vdash (womega\ ((f \wedge more);(g \wedge more))) = (f \wedge more)\ ;\ (womega\ ((g \wedge more);(f \wedge more)))$
proof −
have *1*: $\vdash (f \wedge more)\ ;\ (womega\ ((g \wedge more);(f \wedge more))) \longrightarrow (womega\ ((f \wedge more);(g \wedge more)))$
by (*simp add*: *WOmega-Wagner-2-var*)
have *2*: $\vdash (womega\ ((f \wedge more);(g \wedge more))) = ((f \wedge more);(g \wedge more));\ (womega\ ((f \wedge more);(g \wedge more)))$
by (*meson AndChopA LeftChopImpChop Prop11 WOmegaUnroll WOmega-star-1 WPowerstar-ext lift-imp-trans*)
have *3*: $\vdash (womega\ ((f \wedge more);(g \wedge more))) \longrightarrow (f \wedge more)\ ;\ (womega\ ((g \wedge more);(f \wedge more)))$
by (*metis* (*no-types*, *lifting*) *2 ChopAndB ChopAssoc Prop10 WOmega-Wagner-2-var inteq-reflection*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2*:
$\vdash (omega\ ((f \wedge more)^\frown(g \wedge more))) = (f \wedge more)\ ^\frown\ (omega\ ((g \wedge more)^\frown(f \wedge more)))$
proof −
have *1*: $\vdash (f \wedge more)\ ^\frown\ (omega\ ((g \wedge more)^\frown(f \wedge more))) \longrightarrow$
$(omega\ ((f \wedge more)^\frown(g \wedge more)))$
by (*simp add*: *Omega-Wagner-2-var*)
have *2*: $\vdash (omega\ ((f \wedge more)^\frown(g \wedge more))) = ((f \wedge more)^\frown(g \wedge more))^\frown\ (omega\ ((f \wedge more)^\frown(g \wedge more)))$
by (*metis AndMoreSChopAndMoreEqvAndMoreSChop OmegaUnroll int-eq*)
have *3*: $\vdash (omega\ ((f \wedge more)^\frown(g \wedge more))) \longrightarrow (f \wedge more)\ ^\frown\ (omega\ ((g \wedge more)^\frown(f \wedge more)))$
by (*metis* (*no-types*, *lifting*) *2 Omega-Wagner-2-var RightSChopImpSChop SChopAssoc inteq-reflection*)

show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *WOmega-Wagner-2-var1*:
$\vdash$ (*womega* ((*f*);(*g* $\wedge$ *more*))) = (*f*) ; (*womega* ((*g* $\wedge$ *more*);(*f*)))
proof −
have *1*: $\vdash$ (*f*) ; (*womega* ((*g* $\wedge$ *more*);(*f*))) $\longrightarrow$ (*womega* ((*f*);(*g* $\wedge$ *more*)))
using *WOmega-simulation*[*of f LIFT* (*g* $\wedge$ *more*) ; *f LIFT f* ; (*g* $\wedge$ *more*)]
by (*metis* (*no-types*, *lifting*) *AndChopB ChopAndB ChopAssoc ChopMoreImpMore MoreChopImpMore Prop10 int-iffD1 inteq-reflection lift-imp-trans*)
have *2*: $\vdash$ (*womega* ((*f*);(*g* $\wedge$ *more*))) = ((*f*);(*g* $\wedge$ *more*)); (*womega* ((*f*);(*g* $\wedge$ *more*)))
by (*metis AndChopA Prop11 WOmegaUnroll WOmega-star-1 WPowerstar-inductL-var-equiv inteq-reflection*)
have *25*: $\vdash$ (*g* $\wedge$ *more*);*womega* (*f*;(*g* $\wedge$ *more*)) = ((*g* $\wedge$ *more*);*f* $\wedge$ *more*); ((*g* $\wedge$ *more*);*womega* (*f*;(*g* $\wedge$ *more*)))
by (*metis* (*no-types*, *lifting*) *2 AndChopB ChopAssoc MoreChopImpMore Prop10 int-eq lift-imp-trans*)
have *3*: $\vdash$ (*womega* ((*f*);(*g* $\wedge$ *more*))) $\longrightarrow$ (*f*) ; (*womega* ((*g* $\wedge$ *more*);(*f*)))
using *2 25 WOmega-coinduct-eq-var2*[*of LIFT* (*g* $\wedge$ *more*) ; (*womega* ((*f*);(*g* $\wedge$ *more*))) *LIFT* ((*g* $\wedge$ *more*);(*f*))]
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc RightChopImpChop inteq-reflection*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2-var1*:
$\vdash$ (*omega* ((*f*)⌢(*g* $\wedge$ *more*))) = (*f*) ⌢ (*omega* ((*g* $\wedge$ *more*)⌢(*f*)))
proof −
have *1*: $\vdash$ (*f*) ⌢ (*omega* ((*g* $\wedge$ *more*)⌢(*f*))) $\longrightarrow$ (*omega* ((*f*)⌢(*g* $\wedge$ *more*)))
using *Omega-simulation*[*of f LIFT* (*g* $\wedge$ *more*) ⌢ *f LIFT f* ⌢ (*g* $\wedge$ *more*)]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Prop10 SChopAndB SChopAssoc SChopMoreImpMore int-iffD1 inteq-reflection lift-imp-trans*)
have *2*: $\vdash$ (*omega* ((*f*)⌢(*g* $\wedge$ *more*))) = ((*f*)⌢(*g* $\wedge$ *more*))⌢ (*omega* ((*f*)⌢(*g* $\wedge$ *more*)))
by (*metis AndSChopA OmegaUnroll SChopstarMore-induct-lvar-eq SChopstar-inductL-var-equiv int-eq int-iffI*)
have *3*: $\vdash$ (*omega* ((*f*)⌢(*g* $\wedge$ *more*))) $\longrightarrow$ (*f*) ⌢ (*omega* ((*g* $\wedge$ *more*)⌢(*f*)))
using *2 Omega-coinduct-eq-var2*[*of LIFT* (*g* $\wedge$ *more*) ⌢ (*omega* ((*f*)⌢(*g* $\wedge$ *more*))) *LIFT* ((*g* $\wedge$ *more*)⌢(*f*))]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop RightSChopImpSChop SChopAssoc inteq-reflection*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *WOmega-Wagner-2-var2*:
$\vdash$ (*womega* ((*f* $\wedge$ *more*);(*g*))) = (*f* $\wedge$ *more*) ; (*womega* ((*g*);(*f* $\wedge$ *more*)))
proof −
have *1*: $\vdash$ (*f* $\wedge$ *more*) ; (*womega* ((*g*);(*f* $\wedge$ *more*))) $\longrightarrow$ (*womega* ((*f* $\wedge$ *more*);(*g*)))
by (*metis* (*no-types*, *lifting*) *ChopAssoc WOmega-Wagner-2-var1 WOmega-star-1 WPowerstar-inductL-var-equiv int-eq int-iffD1*)

have *2*: ⊢ (*womega* ((*f* ∧ *more*);(*g*))) = ((*f* ∧ *more*);(*g*)); (*womega* ((*f* ∧ *more*);(*g*)))
by (*metis* (*no-types*, *lifting*) *1 AndChopA ChopAssoc WOmegaUnroll WOmega-Wagner-2-var1 int-iffI inteq-reflection*)
have *3*: ⊢ (*womega* ((*f* ∧ *more*);(*g*))) ⟶ (*f* ∧ *more*) ; (*womega* ((*g*);(*f* ∧ *more*)))
using *1 2 WOmega-Wagner-2-var1*[*of - f*]
by (*metis 1 2 ChopAssoc inteq-reflection*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2-var2*:
⊢ (*omega* ((*f* ∧ *more*)⌢(*g*))) = (*f* ∧ *more*) ⌢ (*omega* ((*g*)⌢(*f* ∧ *more*)))
proof −
have *1*: ⊢ (*f* ∧ *more*) ⌢ (*omega* ((*g*)⌢(*f* ∧ *more*))) ⟶ (*omega* ((*f* ∧ *more*)⌢(*g*)))
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Omega-simulation SChopAndA SChopAssoc int-eq*)
have *2*: ⊢ (*omega* ((*f* ∧ *more*)⌢(*g*))) = ((*f* ∧ *more*)⌢(*g*))⌢ (*omega* ((*f* ∧ *more*)⌢(*g*)))
by (*metis AndMoreSChopAndMoreEqvAndMoreSChop OmegaUnroll inteq-reflection*)
have *3*: ⊢ (*omega* ((*f* ∧ *more*)⌢(*g*))) ⟶ (*f* ∧ *more*) ⌢ (*omega* ((*g*)⌢(*f* ∧ *more*)))
by (*meson 2 Omega-Wagner-2-var1 Prop04 RightSChopEqvSChop SChopAssoc int-iffD1 int-iffD2 int-iffI*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *WOmega-Wagner-3*:
assumes ⊢ ((*f* ∧ *more*) ; (*womega* (*f* ∨ *g*)) ∨ *h*) = (*womega* (*f* ∨ *g*))
shows ⊢ (*womega* (*f* ∨ *g*)) = ((*womega f*) ∨ (*wpowerstar f*);*h*)
proof −
have *1*: ⊢ (*womega* (*f* ∨ *g*)) ⟶ ((*womega f*) ∨ (*wpowerstar* (*f* ∧ *more*));*h*)
using *WOmega-coinduct*[*of LIFT* (*womega* (*f* ∨ *g*)) *h f*] *assms*
by *fastforce*
have *11*: ⊢ (*wpowerstar* (*f* ∧ *more*));*h* = (*wpowerstar f*);*h*
by (*simp add*: *LeftChopEqvChop WPowerstar-more-absorb*)
have *2*: ⊢ (*wpowerstar f*);*h* ⟶ (*womega* (*f* ∨ *g*))
by (*metis ChopImpChop Prop03 WOmega-star-1 WPowerstar-subdist assms inteq-reflection*)
have *3*: ⊢ ((*womega f*) ∨ (*wpowerstar f*);*h*) ⟶ (*womega* (*f* ∨ *g*))
by (*meson 2 Prop02 WOmega-subdist*)
show *?thesis*
using *1 3 int-iffI*
by (*metis 11 inteq-reflection*)
qed

lemma *Omega-Wagner-3*:
assumes ⊢ ((*f* ∧ *more*) ⌢ (*omega* (*f* ∨ *g*)) ∨ (*h*)) = (*omega* (*f* ∨ *g*))
shows ⊢ (*omega* (*f* ∨ *g*)) = ((*omega f*) ∨ (*schopstar f*)⌢(*h*))
proof −
have *1*: ⊢ (*omega* (*f* ∨ *g*)) ⟶ ((*omega f*) ∨ (*schopstar f*)⌢*h*)
using *assms Omega-coinduct*[*of LIFT* (*omega* (*f* ∨ *g*)) *h f*]
by *fastforce*
have *2*: ⊢ (*schopstar f*)⌢*h* ⟶ (*omega* (*f* ∨ *g*))

by (*meson Omega-star-1 Prop03 Prop11 SChopImpSChop SChopstar-subdist assms lift-imp-trans*)
have *3*: $\vdash ((omega\ f) \vee (schopstar\ f)^\frown h) \longrightarrow (omega\ (f \vee g))$
by (*meson 2 Prop02 Omega-subdist*)
show *?thesis* **using** *1 3* **by** *fastforce*
qed

lemma *WOmega-Wagner-1-var*:
$\vdash (womega\ ((wpowerstar\ f);(f \wedge more))) = (womega\ f)$
by (*metis WOmega-More-absorb WOmega-Wagner-1 WOmega-Wagner-2-var1 WOmega-star-1 WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)

lemma *Omega-Wagner-1-var3*:
$\vdash (omega\ ((schopstar\ f)^\frown(f \wedge more))) = (omega\ f)$
by (*metis Omega-Wagner-1-var2 Omega-Wagner-2-var1 Omega-star-1 inteq-reflection*)

lemma *WOmega-star-4*:
$\vdash (wpowerstar\ (womega\ f)) = (empty \vee (womega\ f))$
by (*simp add*: *WOmega-1 WPowerstar-boffa-var*)

lemma *WOmega-star-5*:
$\vdash (womega\ f);(wpowerstar\ (womega\ f)) = (womega\ f)$
proof −
have *1*: $\vdash (womega\ f);(wpowerstar\ (womega\ f)) \longrightarrow (womega\ f)$
by (*simp add*: *WOmega-1*)
have *2*: $\vdash (womega\ f) \longrightarrow (womega\ f);(wpowerstar\ (womega\ f))$
by (*simp add*: *WOmega-sup-id WPowerstar-imp-empty int-iffD2*)
show *?thesis*
by (*simp add*: *1 2 int-iffI*)
qed

lemma *WOmega-or-unfold*:
$\vdash ((womega\ f) \vee ((wpowerstar\ f);(g \wedge more));\ (womega\ (f \vee g))) = (womega\ (f \vee g))$
proof −
have *10*: $\vdash (womega\ (f \vee g)) = ((f \vee g) \wedge more);(womega\ (f \vee g))$
by (*simp add*: *WOmegaUnroll*)
have *11*: $\vdash ((f \vee g) \wedge more) = ((f \wedge more) \vee (g \wedge more))$
by *fastforce*
have *1*: $\vdash (womega\ (f \vee g)) = ((f \wedge more);(womega\ (f \vee g)) \vee (g \wedge more);(womega\ (f \vee g)))$
using *10 11* **by** (*metis OrChopEqv inteq-reflection*)
have *2*: $\vdash((wpowerstar\ f);(g \wedge more));\ (womega\ (f \vee g)) = (wpowerstar\ f);((g \wedge more);\ (womega\ (f \vee g)))$
by (*meson ChopAssoc int-iffD1 int-iffD2 int-iffI*)
show *?thesis* **using** *1 2 WOmega-Wagner-3*[*of f g LIFT* $(g \wedge more);(womega\ (f \vee g))$]
by (*metis int-eq*)
qed

lemma *Omega-or-unfold*:
$\vdash ((omega\ f) \vee ((schopstar\ f)^\frown(g \wedge more))^\frown\ (omega\ (f \vee g))) = (omega\ (f \vee g))$

proof −
have *10*: ⊢ (*omega* (*f* ∨ *g*)) = ((*f* ∨ *g*) ∧ *more*)⌢(*omega* (*f* ∨ *g*))
by (*simp add*: *OmegaUnroll*)
have *11*: ⊢ ((*f* ∨ *g*) ∧ *more*) = ((*f* ∧ *more*) ∨ (*g* ∧ *more*))
by *fastforce*
have *1*: ⊢ (*omega* (*f* ∨ *g*)) = ((*f* ∧ *more*)⌢(*omega* (*f* ∨ *g*)) ∨ (*g* ∧ *more*)⌢(*omega* (*f* ∨ *g*)))
using *10 11* **by** (*metis OrSChopEqv inteq-reflection*)
have *2*: ⊢((*schopstar f*)⌢(*g* ∧ *more*))⌢ (*omega* (*f* ∨ *g*)) = (*schopstar f*)⌢((*g* ∧ *more*)⌢ (*omega* (*f* ∨ *g*)))
by (*meson SChopAssoc int-iffD1 int-iffD2 int-iffI*)
show *?thesis* **using** *1 2 Omega-Wagner-3*[*of f g LIFT* (*g* ∧ *more*)⌢(*omega* (*f* ∨ *g*))]
by (*metis int-eq*)
qed

lemma *WOmega-or-unfold-coinduct*:
⊢ (*womega* (*f* ∨ *g*)) ⟶ (*womega* (((*wpowerstar f*);(*g* ∧ *more*))))
∨ (*wpowerstar* ((*wpowerstar f*);(*g* ∧ *more*)));(*womega f*)
using *WOmega-or-unfold*[*of f g*]
WOmega-coinduct-eq[*of LIFT* (*womega* (*f* ∨ *g*)) *LIFT* (*womega f*)
LIFT (*wpowerstar f*);(*g* ∧ *more*)]
by (*metis Prop10 Prop11 Prop12 RightChopImpMoreRule inteq-reflection lift-and-com*)

lemma *Omega-or-unfold-coinduct*:
⊢ (*omega* (*f* ∨ *g*)) ⟶ (*omega* (((*schopstar f*)⌢(*g* ∧ *more*))))
∨ (*schopstar* ((*schopstar f*)⌢(*g* ∧ *more*)))⌢(*omega f*)
using *Omega-or-unfold*[*of f g*]
Omega-coinduct-eq[*of LIFT* (*omega* (*f* ∨ *g*)) *LIFT* (*omega f*)
LIFT (*schopstar f*)⌢(*g* ∧ *more*)]
by (*metis Prop10 Prop12 RightSChopImpMoreRule int-eq int-iffD2 lift-and-com*)

lemma *WOmega-or-unfold-induct*:
⊢ (*wpowerstar* ((*wpowerstar f*);(*g* ∧ *more*)));(*womega f*) ⟶ (*womega* (*f* ∨ *g*))
using *WOmega-or-unfold*[*of f g*]
using *WPowerstar-induct-leq* **by** *blast*

lemma *Omega-or-unfold-induct*:
⊢ (*schopstar* ((*schopstar f*)⌢(*g* ∧ *more*)))⌢(*omega f*) ⟶ (*omega* (*f* ∨ *g*))
using *Omega-or-unfold*[*of f g*]
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-leq inteq-reflection lift-imp-trans schop-d-def*)

lemma *WOmega-Wagner-1-gen*:
⊢ (*womega* ((*f* ∧ *more*);(*wpowerstar g*))) ⟶ (*womega* (*f* ∨ *g*))
proof −
have *01*: ⊢ *f* ⟶ (*f* ∨ *g*)
by *auto*
have *02*: ⊢ (*wpowerstar g*) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*metis WPowerstar-subdist WPowerstar-swap inteq-reflection*)
have *03*: ⊢ ((*f* ∧ *more*);(*wpowerstar g*)) ⟶ ((*f* ∨ *g*) ∧ *more*);(*wpowerstar* (*f* ∨ *g*))
using *01 02 ChopImpChop*
by (*metis Prop10 Prop12 int-iffD1 inteq-reflection lift-and-com*)

have *1*: $\vdash$ (*womega* ((*f* $\wedge$ *more*);(*wpowerstar g*))) $\longrightarrow$
(*womega* (((*f* $\vee$ *g*) $\wedge$ *more*);(*wpowerstar* (*f* $\vee$ *g*))))
by (*simp add*: *03 WOmega-iso*)
show *?thesis* **using** *WOmega-Wagner-1*
by (*metis 03 Prop12 WOmega-iso WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)
qed

lemma *Omega-Wagner-1-gen*:
$\vdash$ (*omega* ((*f* $\wedge$ *more*)$\frown$(*schopstar g*))) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))

proof −
have *1*: $\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite* $\vee$ (*g* $\wedge$ *more*) $\wedge$ *finite*) =(((*f* $\vee$ *g*) $\wedge$ *more*) $\wedge$ *finite*)
by *fastforce*
show *?thesis*
using *Omega-WOmega*[*of LIFT* ((*f* $\wedge$ *more*)$\frown$(*schopstar g*))] *Omega-WOmega*[*of LIFT* (*f* $\vee$ *g*)]
WOmega-Wagner-1-gen[*of LIFT* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *LIFT* ((*g* $\wedge$ *more*) $\wedge$ *finite*)]
SChopstar-WPowerstar[*of LIFT* (*g* $\wedge$ *more*)] *SChopstar-WPowerstar-more*[*of g*] *1*
by (*metis* (*no-types*, *lifting*) *Omega-Wagner-1 Prop10 Prop12 SChopImpFinite SChopstar-WPowerstar-more SChopstar-finite int-iffD1 inteq-reflection itl-def*(*9*) *lift-and-com omega-d-def*)
qed

lemma *WOmega-swap*:
$\vdash$ (*womega* (*f* $\vee$ *g*)) = (*womega* (*g* $\vee$ *f*))
proof −
have *1*: $\vdash$ (*f* $\vee$ *g*) = (*g* $\vee$ *f*)
by *fastforce*
show *?thesis*
by (*metis 1 WOmega-star-5 int-eq*)
qed

lemma *Omega-swap*:
$\vdash$ (*omega* (*f* $\vee$ *g*)) = (*omega* (*g* $\vee$ *f*))
proof −
have *1*: $\vdash$ (*f* $\vee$ *g*) = (*g* $\vee$ *f*)
by *fastforce*
show *?thesis*
by (*metis 1 Omega-star-1 inteq-reflection*)
qed

lemma *WOmega-Wagner-1-var-gen*:
$\vdash$ (*womega* ((*wpowerstar f*);(*g* $\wedge$ *more*))) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
proof −
have *1*: $\vdash$ (*womega* ((*wpowerstar f*);(*g* $\wedge$ *more*))) =
(*wpowerstar f*);(*womega* ((*g* $\wedge$ *more*);(*wpowerstar f*)))
by (*simp add*: *WOmega-Wagner-2-var1*)
have *2*: $\vdash$ (*womega* ((*g* $\wedge$ *more*);(*wpowerstar f*))) $\longrightarrow$ (*womega* (*g* $\vee$ *f*))
using *WOmega-Wagner-1-gen*[*of g f*] **by** *blast*
have *5*: $\vdash$ (*wpowerstar f*);(*womega* ((*g* $\wedge$ *more*);(*wpowerstar f*)))
$\longrightarrow$ (*wpowerstar f*);(*womega* (*f* $\vee$ *g*))
by (*metis 2 RightChopImpChop WOmega-swap int-eq*)

have *6*: $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*wpowerstar* (*f* $\vee$ *g*))
by (*simp add*: *WPowerstar-subdist*)
have *7*: $\vdash$ (*wpowerstar f*);(*womega* (*f* $\vee$ *g*)) $\longrightarrow$ (*wpowerstar* (*f* $\vee$ *g*));(*womega* (*f* $\vee$ *g*))
by (*simp add*: *6 LeftChopImpChop*)
have *8*: $\vdash$ (*wpowerstar* (*f* $\vee$ *g*));(*womega* (*f* $\vee$ *g*)) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*simp add*: *WOmega-star-1 int-iffD1*)
show *?thesis*
using *1 5 7 8*
by (*metis int-eq lift-imp-trans*)
qed

lemma *Omega-Wagner-1-var-gen*:
$\vdash$ (*omega* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*))) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
proof −
have *1*: $\vdash$ (*omega* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*))) =
(*schopstar f*)$\frown$(*omega* ((*g* $\wedge$ *more*)$\frown$(*schopstar f*)))
by (*simp add*: *Omega-Wagner-2-var1*)
have *2*: $\vdash$ (*omega* ((*g* $\wedge$ *more*)$\frown$(*schopstar f*))) $\longrightarrow$ (*omega* (*g* $\vee$ *f*))
using *Omega-Wagner-1-gen*[*of g f*] **by** *blast*
have *3*: $\vdash$ (*g* $\vee$ *f*) = (*f* $\vee$ *g*)
by *fastforce*
have *4*: $\vdash$ (*omega* (*g* $\vee$ *f*)) = (*omega* (*f* $\vee$ *g*))
by (*simp add*: *Omega-swap*)
have *5*: $\vdash$ (*schopstar f*)$\frown$(*omega* ((*g* $\wedge$ *more*)$\frown$(*schopstar f*)))
$\longrightarrow$ (*schopstar f*)$\frown$(*omega* (*f* $\vee$ *g*))
by (*metis 2 3 RightSChopImpSChop inteq-reflection*)
have *6*: $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*metis 3 MP Prop03 SBaGen SBaSCSImpSCS int-eq*)
have *7*: $\vdash$ (*schopstar f*)$\frown$(*omega* (*f* $\vee$ *g*)) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))$\frown$(*omega* (*f* $\vee$ *g*))
by (*simp add*: *6 LeftSChopImpSChop*)
have *8*: $\vdash$ (*schopstar* (*f* $\vee$ *g*))$\frown$(*omega* (*f* $\vee$ *g*)) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
by (*simp add*: *Omega-star-1 int-iffD1*)
show *?thesis*
using *1 5 7 8* **by** *fastforce*
qed

lemma *WOmega-Denest*:
$\vdash$ (*womega* (*f* $\vee$ *g*)) =
((*womega* ((*wpowerstar f*);(*g* $\wedge$ *more*))) $\vee$
(*wpowerstar* ((*wpowerstar f*);(*g* $\wedge$ *more*)));(*womega f*))
proof −
have *1*: $\vdash$ (*womega* (*f* $\vee$ *g*)) $\longrightarrow$ (*womega* (((*wpowerstar f*);(*g* $\wedge$ *more*))))
$\vee$ (*wpowerstar* ((*wpowerstar f*);(*g* $\wedge$ *more*)));(*womega f*)
by (*simp add*: *WOmega-or-unfold-coinduct*)
have *2*: $\vdash$ (*womega* ((*wpowerstar f*);(*g* $\wedge$ *more*))) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*simp add*: *WOmega-Wagner-1-var-gen*)
have *3*: $\vdash$ (*wpowerstar* ((*wpowerstar f*);(*g* $\wedge$ *more*)));(*womega f*) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*simp add*: *WOmega-or-unfold-induct*)
show *?thesis*
by (*meson 1 2 3 Prop02 int-iffI*)

qed

lemma *Omega-Denest*:
$\vdash$ (*omega* (*f* $\vee$ *g*)) =
((*omega* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*))) $\vee$
(*schopstar* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*)))$\frown$(*omega f*))
proof −
have *1*: $\vdash$ (*omega* (*f* $\vee$ *g*)) $\longrightarrow$ (*omega* (((*schopstar f*)$\frown$(*g* $\wedge$ *more*))))
$\vee$ (*schopstar* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*)))$\frown$(*omega f*)
by (*simp add*: *Omega-or-unfold-coinduct*)
have *2*: $\vdash$ (*omega* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*))) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
by (*simp add*: *Omega-Wagner-1-var-gen*)
have *3*: $\vdash$ (*schopstar* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*)))$\frown$(*omega f*) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
by (*simp add*: *Omega-or-unfold-induct*)
show *?thesis*
by (*meson 1 2 3 Prop02 int-iffI*)
qed

lemma *WOmega-or-refine*:
assumes $\vdash$ (*g* $\wedge$ *more*); (*f* $\wedge$ *more*) $\longrightarrow$ (*f* $\wedge$ *more*) ; (*wpowerstar* (*f* $\vee$ *g*))
shows $\vdash$ (*womega* (*f* $\vee$ *g*)) = ((*womega f*) $\vee$ (*wpowerstar f*);(*womega g*))
proof −
have *1*: $\vdash$ (*wpowerstar g*);(*f* $\wedge$ *more*) $\longrightarrow$ (*f* $\wedge$ *more*);(*wpowerstar* (*f* $\vee$ *g*))
using *assms WPowerstar-invol WPowerstar-sim1*[*of LIFT* (*g* $\wedge$ *more*) *LIFT* (*f* $\wedge$ *more*) *LIFT powerstar* (*f* $\vee$ *g*)]

by (*metis* (*no-types*, *opaque-lifting*) *ChopstarEqvPowerstar Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have *2*: $\vdash$ (*womega* (*f* $\vee$ *g*)) = ((*womega g*) $\vee$ ((*wpowerstar g*);(*f* $\wedge$ *more*));(*womega* (*f* $\vee$ *g*)))
by (*metis WOmega-swap WOmega-or-unfold inteq-reflection*)
have *3*: $\vdash$ ((*womega g*) $\vee$ ((*wpowerstar g*);(*f* $\wedge$ *more*));(*womega* (*f* $\vee$ *g*))) $\longrightarrow$
(*womega g*) $\vee$ ((*f* $\wedge$ *more*);(*wpowerstar* (*f* $\vee$ *g*)));(*womega* (*f* $\vee$ *g*))
by (*metis 1 2 AndChopB Prop08 Prop10 int-iffD2 inteq-reflection*)
have *40*: $\vdash$ (*womega g*) $\longrightarrow$ (*f* $\wedge$ *more*);(*womega* (*f* $\vee$ *g*)) $\vee$ (*womega g*)
by (*simp add*: *Prop05*)
have *41*: $\vdash$ ((*f* $\wedge$ *more*);(*wpowerstar* (*f* $\vee$ *g*)));(*womega* (*f* $\vee$ *g*)) $\longrightarrow$
(*f* $\wedge$ *more*);(*womega* (*f* $\vee$ *g*)) $\vee$ (*womega g*)
by (*metis* (*mono-tags*, *lifting*) *ChopAssoc WOmega-star-1 intI inteq-reflection unl-lift2*)
have *4*: $\vdash$ (*womega g*) $\vee$ ((*f* $\wedge$ *more*);(*wpowerstar* (*f* $\vee$ *g*)));(*womega* (*f* $\vee$ *g*)) $\longrightarrow$
(*f* $\wedge$ *more*);(*womega* (*f* $\vee$ *g*)) $\vee$ (*womega g*)
using *40 41 Prop02* **by** *blast*
have *5*: $\vdash$ (*womega* (*f* $\vee$ *g*)) $\longrightarrow$ ((*womega f*) $\vee$ (*wpowerstar f*);(*womega g*))
by (*metis 2 3 ChopAssoc WOmega-coinduct-var WOmega-star-1 inteq-reflection*)
have *6*: $\vdash$ ((*womega f*) $\vee$ (*wpowerstar f*);(*womega g*)) $\longrightarrow$
(*womega* (*f* $\vee$ *g*)) $\vee$ (*wpowerstar* (*f* $\vee$ *g*));(*womega* (*f* $\vee$ *g*))
by (*metis ChopImpChop Prop02 Prop05 WOmega-star-1 WOmega-subdist WOmega-swap WPowerstar-subdist int-eq*)
have *7*: $\vdash$ ((*womega f*) $\vee$ (*wpowerstar f*);(*womega g*)) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
using *6 WOmega-star-1*
by (*metis Prop02 int-iffD1 inteq-reflection lift-imp-trans*)

show *?thesis*
using *5 7 int-iffI* **by** *blast*
qed

lemma *Omega-or-refine*:
assumes $\vdash (g \wedge more)^\frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more)^\frown (schopstar\ (f \vee g))$
shows $\vdash (omega\ (f \vee g)) = ((omega\ f) \vee (schopstar\ f)^\frown (omega\ g))$
proof −
have *1*: $\vdash (schopstar\ (g\ ))^\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more)^\frown(schopstar\ (f \vee g))$
using *assms*
using *SChopstar-QuasicommMore-var* **by** *blast*
have *2*: $\vdash (omega\ (f \vee g)) = ((omega\ g) \vee ((schopstar\ g)^\frown(f \wedge more))^\frown(omega\ (f \vee g)))$
by (*metis Omega-swap Omega-or-unfold inteq-reflection*)
have *20*: $\vdash ((schopstar\ g)^\frown(f \wedge more))^\frown(omega\ (f \vee g)) =$
$((schopstar\ g)^\frown((f \wedge more) \wedge finite))^\frown(omega\ (f \vee g))$
by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore Prop10 Prop12 SChopAssoc int-eq int-iffD2 itl-def*(*9*))
have *25*: $\vdash ((schopstar\ g)^\frown(f \wedge more))^\frown(omega\ (f \vee g)) \longrightarrow$
$((f \wedge more)^\frown(schopstar\ (f \vee g)))^\frown(omega\ (f \vee g))$
using *1 20* **by** (*metis LeftSChopImpSChop inteq-reflection*)
have *3*: $\vdash ((omega\ g) \vee ((schopstar\ g)^\frown(f \wedge more))^\frown(omega\ (f \vee g))) \longrightarrow$
$(omega\ g) \vee ((f \wedge more)^\frown(schopstar\ (f \vee g)))^\frown(omega\ (f \vee g))$
using *25* **by** *auto*
have *4*: $\vdash (omega\ g) \vee ((f \wedge more)^\frown(schopstar\ (f \vee g)))^\frown(omega\ (f \vee g)) \longrightarrow$
$(omega\ g) \vee\ (f \wedge more)^\frown(omega\ (f \vee g))$
by (*metis* (*mono-tags*, *lifting*) *Omega-star-1 SChopAssoc intI int-eq intensional-rews*(*3*))
have *5*: $\vdash (omega\ (f \vee g)) \longrightarrow ((omega\ f) \vee (schopstar\ f)^\frown(omega\ g))$
by (*metis 2 3 Omega-coinduct Omega-star-1 SChopAssoc inteq-reflection*)
have *50*: $\vdash (omega\ f) \longrightarrow (omega\ (f \vee g))$
by (*simp add*: *Omega-subdist*)
have *51*: $\vdash (omega\ g) \longrightarrow (omega\ (f \vee g))$
by (*metis Omega-swap Omega-subdist inteq-reflection*)
have *52*: $\vdash (schopstar\ f) \longrightarrow (schopstar\ (f \vee g))$
by (*simp add*: *SChopstar-subdist*)
have *6*: $\vdash ((omega\ f) \vee (schopstar\ f)^\frown(omega\ g)) \longrightarrow$
$(omega\ (f \vee g)) \vee (schopstar\ (f \vee g))^\frown(omega\ (f \vee g))$
by (*metis 50 51 52 Omega-star-1 Prop02 Prop05 SChopImpSChop inteq-reflection*)
have *7*: $\vdash ((omega\ f) \vee (schopstar\ f)^\frown(omega\ g)) \longrightarrow (omega\ (f \vee g))$
using *6 Omega-star-1* **by** *fastforce*
show *?thesis*
using *5 7 int-iffI* **by** *blast*
qed

lemma *WOmega-bachmair-dershowitz*:
assumes $\vdash (g \wedge more);(f \wedge more) \longrightarrow (f \wedge more);(wpowerstar\ (f \vee g))$
shows $(\vdash (womega\ (f \vee g)) = \#False) = (\vdash ((womega\ f) \vee (womega\ g)) = \#False)$
proof −

have *1*: $(\vdash (womega\ (f \vee g)) = \#False) \Longrightarrow (\vdash ((womega\ f) \vee (womega\ g)) = \#False)$
using *assms*
by (*metis Prop10 WOmega-subdist-var int-eq int-simps*(*18*))
have *2*: $(\vdash ((womega\ f) \vee (womega\ g)) = \#False) \implies (\vdash (womega\ (f \vee g)) = \#False)$
using *assms WOmega-or-refine*[*of g f*]
by (*metis Prop03 Prop10 WOmega-star-1 int-simps*(*18*) *int-simps*(*25*) *int-simps*(*26*) *inteq-reflection*)
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *Omega-bachmair-dershowitz*:
assumes $\vdash (g \wedge more)^\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more)^\frown(schopstar\ (f \vee g))$
shows $(\vdash (omega\ (f \vee g)) = \#False) = (\vdash ((omega\ f) \vee (omega\ g)) = \#False)$
proof −
have *1*: $(\vdash (omega\ (f \vee g)) = \#False) \Longrightarrow (\vdash ((omega\ f) \vee (omega\ g)) = \#False)$
using *assms*
by (*metis Prop10 Omega-subdist-var int-eq int-simps*(*18*))
have *2*: $(\vdash ((omega\ f) \vee (omega\ g)) = \#False) \implies (\vdash (omega\ (f \vee g)) = \#False)$
using *assms Omega-or-refine*[*of g f*]
by (*metis Prop03 SChopRightFalse int-simps*(*14*) *int-simps*(*16*) *int-simps*(*21*) *int-simps*(*9*) *inteq-reflection*)
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *Omega-and-more-imp-more*:
$\vdash (omega\ (f \wedge more)) \longrightarrow more$
by (*metis AndSChopB MoreSChopImpMore OmegaUnroll inteq-reflection lift-imp-trans*)

lemma *WOmega-and-more-imp-more*:
$\vdash (womega\ (f \wedge more)) \longrightarrow more$
by (*metis ChopImpDi DiAndB DiMoreEqvMore WOmegaUnroll int-eq lift-imp-trans*)

lemma *Omega-and-fmore-imp-more*:
$\vdash (omega\ ((f \wedge more) \wedge finite)) \longrightarrow more$
by (*metis Omega-and-more-imp-more Omega-subdist OrFiniteInf inteq-reflection lift-imp-trans*)

lemma *WOmega-and-fmore-imp-more*:
$\vdash (womega\ ((f \wedge more) \wedge finite)) \longrightarrow more$
by (*metis LeftChopImpMoreRule Prop12 WOmegaUnroll int-iffD1 inteq-reflection lift-and-com*)

lemma *womega-len*:
$\vdash womega\ skip = (len\ k);womega\ skip$
proof (*induct k*)
case *0*
then show *?case*
by (*metis EmptyChop inteq-reflection wpow-0 wpower-len*)
next

case (*Suc k*)
then show *?case*
proof −
have *1*: ⊢ *len Suc k = len k* ; *skip*
by (*metis LenNPlusOneB Suc-eq-plus1*)
have *2*: ⊢ *len Suc k* ; *womega skip = len k*;(*skip*;*womega skip*)
by (*metis 1 ChopAssoc inteq-reflection*)
have *3*: ⊢ *skip*;*womega skip = womega skip*
by (*metis AndChopA Prop11 WOmegaUnroll WPowerstar-inductL-var-equiv WPowerstar-more-absorb inteq-reflection*)
show *?thesis*
by (*metis 2 3 Suc inteq-reflection*)
qed
qed

lemma *omega-len*:
⊢ *omega skip* = (*len k*)⌢*omega skip*
proof (*induct k*)
case *0*
then show *?case*
by (*metis EmptyChop FiniteAndEmptyEqvEmpty inteq-reflection lift-and-com schop-d-def wpow-0 wpower-len*)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: ⊢ *len Suc k = len k* ⌢ *skip*
by (*simp add*: *len-k-schop*)
have *2*: ⊢ *len Suc k* ⌢ *omega skip = len k*⌢(*skip*⌢*omega skip*)
by (*metis 1 SChopAssoc inteq-reflection*)
have *3*: ⊢ *skip*⌢*omega skip = omega skip*
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite OmegaUnroll Omega-Wagner-1 Omega-star-1 SChopAssoc inteq-reflection schopstar-skip-finite schopstar-skip-fmore*)
show *?thesis*
by (*metis 2 3 Suc inteq-reflection*)
qed
qed

lemma *womega-exist-len*:
⊢ (*womega skip*) = (∃ *k*. *len k*; (*womega skip*))
proof −
have *1*: ⊢ (*womega skip*) = (*wpowerstar skip*);(*womega skip*)
by (*meson Prop11 WOmega-star-1*)
have *2*: ⊢ (*wpowerstar skip*) = (∃ *k*. *len k*)
unfolding *wpowerstar-d-def*
by (*simp add*: *ExEqvRule wpower-len*)
have *3*: ⊢ (∃ *k*. *len k*);(*womega skip*) = (∃ *k*. *len k*; (*womega skip*))
by (*metis ExistChop int-eq*)
show *?thesis* **using** *1 2 3*

by (*metis int-eq*)
qed

lemma *omega-exist-len*:
$\vdash$ (*omega skip*) = ($\exists$ *k*. *len k*$\frown$ (*omega skip*))
proof −
have *1*: $\vdash$ (*omega skip*) = (*schopstar skip*)$\frown$(*omega skip*)
by (*meson Omega-star-1 Prop11*)
have *2*: $\vdash$ (*schopstar skip*) = ($\exists$ *k*. *len k*)
by (*metis ExEqvRule int-eq schopstar-skip-finite wpower-len wpowerstar-d-def wpowerstar-skip-finite*)
have *3*: $\vdash$ ($\exists$ *k*. *len k*)$\frown$(*omega skip*) = ($\exists$ *k*. *len k*$\frown$ (*omega skip*))
by (*metis ExistSChop inteq-reflection*)
show *?thesis* **using** *1 2 3*
by (*metis int-eq*)
qed

lemma *not-len-and-womega*:
$\vdash$ ¬ (*len k* ∧ (*womega skip*))
using *womega-len*[*of Suc k*]
using *len-len-suc-not* **by** *fastforce*

lemma *not-len-and-omega*:
$\vdash$ ¬ (*len k* ∧ (*omega skip*))
using *omega-len*[*of Suc k*]
len-len-suc-not-schop **by** *fastforce*

lemma *not-len-and-womega-var*:
$\vdash$ (*womega skip*) ⟶ ¬(*len k*)
using *not-len-and-womega* **by** *fastforce*

lemma *not-len-and-omega-var*:
$\vdash$ (*omega skip*) ⟶ ¬(*len k*)
using *not-len-and-omega* **by** *fastforce*

lemma *womega-skip-inf*:
$\vdash$ (*womega skip*) ⟶ *inf*
proof −
have *1*: $\vdash$ *finite* = ($\exists$ *k*. *len k*)
by (*simp add*: *intI itl-defs len-defs nfinite-conv-nlength-enat*)
have *2*: $\vdash$ *inf* = (∀ *k*. ¬ *len k*)
using *1* **unfolding** *finite-d-def* **by** *fastforce*
have *3*: $\vdash$ (*womega skip*) ⟶ (∀ *k*. ¬ *len k*)
using *not-len-and-womega-var* **by** *fastforce*
show *?thesis* **using** *2 3*
by (*metis int-eq*)
qed

lemma *omega-skip-inf*:

$\vdash$ (*omega skip*) $\longrightarrow$ *inf*
proof −
have *1*: $\vdash$ *finite* = ($\exists$ *k*. *len k*)
by (*simp add*: *intI itl-defs len-defs nfinite-conv-nlength-enat*)
have *2*: $\vdash$ *inf* = ($\forall$ *k*. $\neg$ *len k*)
using *1* **unfolding** *finite-d-def* **by** *fastforce*
have *3*: $\vdash$ (*omega skip*) $\longrightarrow$ ($\forall$ *k*. $\neg$ *len k*)
using *not-len-and-omega-var* **by** *fastforce*
show *?thesis* **using** *2 3*
by *fastforce*
qed

lemma *womega-fmore-inf*:
$\vdash$ (*womega fmore*) $\longrightarrow$ *inf*
using *wpowerstar-skip-fmore*
by (*metis SkipChopFiniteImpFinite WOmega-Wagner-1 WOmega-iso inteq-reflection lift-imp-trans womega-skip-inf wpowerstar-skip-finite*)

lemma *omega-fmore-inf*:
$\vdash$ (*omega fmore*) $\longrightarrow$ *inf*
by (*metis AndChopA ChopEmpty WOmega-iso inteq-reflection lift-imp-trans omega-d-def womega-fmore-inf*)

lemma *womega-and-fmore-inf*:
$\vdash$ (*womega* (*f* $\wedge$ *fmore*)) $\longrightarrow$ *inf*
by (*meson Prop12 WOmega-iso int-iffD1 lift-and-com lift-imp-trans womega-fmore-inf*)

lemma *omega-and-fmore-inf*:
$\vdash$ (*omega* (*f* $\wedge$ *fmore*)) $\longrightarrow$ *inf*
by (*meson Prop12 Omega-iso int-iffD1 lift-and-com lift-imp-trans omega-fmore-inf*)

lemma *womega-and-empty*:
$\vdash$ $\neg$(*womega* (*f* $\wedge$ *empty*))
by (*metis WOmega-Wagner-1 WOmega-empty-zero WPowerstar-and-empty inteq-reflection*)

lemma *omega-and-empty*:
$\vdash$ $\neg$(*omega* (*f* $\wedge$ *empty*))
by (*meson MP Omega-empty-zero Omega-iso Prop11 Prop12 lift-and-com lift-imp-neg*)

lemma *womega-split-empty-more*:
$\vdash$ (*womega f*) = (*womega* (*f* $\wedge$ *more*))
by (*metis WOmega-Wagner-1 WPowerstar-more-absorb inteq-reflection*)

lemma *omega-split-empty-more*:
$\vdash$ (*omega f*) = ((*omega* (*f* $\wedge$ *more*)))
by (*metis Omega-Wagner-1 SChopstar-and-more inteq-reflection*)

lemma *omega-split-empty-more-var*:
$\vdash$ (*omega* (*f* $\wedge$ *more*)) = (*omega* (*f* $\wedge$ *fmore*))
by (*metis AndMoreSChopEqvAndFmoreChop ChopEmpty EmptySChop Omega-Wagner-2-var1 inteq-reflection*)

lemma *omega-imp-inf*:
$\vdash$ (*omega f*) $\longrightarrow$ *inf*
by (*metis int-eq omega-and-fmore-inf omega-split-empty-more omega-split-empty-more-var*)

lemma *inf-imp-omega-skip*:
$\vdash$ *inf* $\longrightarrow$ (*omega skip*)
proof −
have *1*: $\vdash$ ((*skip* $\wedge$ *more*) $\wedge$ *finite*) = *skip*
by (*metis DiIntro DiSkipEqvMore Prop10 WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
have *2*: $\vdash$ *inf* $\longrightarrow$ *skip*;*inf*
by (*metis AndInfEqvChopFalse ChopAndInf MoreAndInfEqvInf MoreEqvSkipChopTrue infinite-d-def int-eq int-iffD1*)
have *3*: $\vdash$ *inf* $\longrightarrow$ ((*skip* $\wedge$ *more*) $\wedge$ *finite*);*inf*
using *1 2* **by** (*metis int-eq*)
show *?thesis*
by (*simp add*: *3 OmegaWeakCoinduct schop-d-def*)
qed

lemma *Omega-schopstar-max-element*:
$\vdash$ *f* $\longrightarrow$ (*omega skip*) $\vee$ (*schopstar skip*)
proof −
have *1*: $\vdash$ (*schopstar skip*) = *finite*
by (*meson Prop11 schopstar-skip-finite*)
have *2*: $\vdash$ (*omega skip*) = *inf*
by (*simp add*: *inf-imp-omega-skip int-iffI omega-skip-inf*)
show *?thesis* **using** *1 2* **unfolding** *finite-d-def* **by** *fastforce*
qed

lemma *Omega-fmore-absorb*:
$\vdash$ (*omega f*) = (*omega* (*f* $\wedge$ *fmore*))
by (*metis int-eq omega-split-empty-more omega-split-empty-more-var*)

lemma *Omega-top*:
$\vdash$ (*omega f*);(*omega empty*) = (*omega f*)
proof −
have *1*: $\vdash$ (*omega f*) $\longrightarrow$ *inf*
by (*simp add*: *omega-imp-inf*)
have *2*: $\vdash$ (*omega f*);(*omega empty*) $\longrightarrow$ (*omega f*)

by (*simp add*: *Omega-1*)
have *3*: $\vdash$ (*omega f*) $\longrightarrow$ (*omega f*);(*omega empty*)
by (*metis 1 AndInfChopEqvAndInf Prop10 int-iffD2 inteq-reflection*)
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *womega-and-inf*:
$\vdash$ (*womega* (*f* $\wedge$ *inf*)) = (*f* $\wedge$ *inf*)
by (*metis* (*no-types*, *lifting*) *AndInfChopEqvAndInf AndMoreAndInfEqvAndInf Prop10 Prop12 WOmegaUnroll int-iffD1 inteq-reflection*)

lemma *womega-split-finite-inf*:
$\vdash$ (*womega f*) =
(*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$ (*womega* (*f* $\wedge$ *finite*)))
proof −
have *0*: $\vdash$ (*womega f*) = (*womega* ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*)))
by (*metis OrFiniteInf int-eq*)
have *1*: $\vdash$ (*womega* ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*))) =
(*womega* (*wpowerstar* (*f* $\wedge$ *finite*);((*f* $\wedge$ *inf*) $\wedge$ *more*)) $\vee$
wpowerstar (*wpowerstar* (*f* $\wedge$ *finite*);((*f* $\wedge$ *inf*) $\wedge$ *more*));*womega* (*f* $\wedge$ *finite*))
using *WOmega-Denest*[*of LIFT f* $\wedge$ *finite LIFT* (*f* $\wedge$ *inf*)]
by *blast*
have *15*: $\vdash$ ((*f* $\wedge$ *inf*) $\wedge$ *more*) = (*f* $\wedge$ *inf*)
using *MoreAndInfEqvInf* **by** *auto*
have *2*: $\vdash$ (*womega* (*wpowerstar* (*f* $\wedge$ *finite*);((*f* $\wedge$ *inf*) $\wedge$ *more*))) = *wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*metis 15 AndInfChopEqvAndInf WOmega-Wagner-2-var1 inteq-reflection*)
have *3*: $\vdash$ *wpowerstar* (*wpowerstar* (*f* $\wedge$ *finite*);((*f* $\wedge$ *inf*) $\wedge$ *more*)) =
(*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))
by (*metis 15 2 WOmega-star-4 inteq-reflection*)
have *4*: $\vdash$ (*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*))); *womega* (*f* $\wedge$ *finite*) =
(*womega* (*f* $\wedge$ *finite*) $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))
by (*meson AndInfChopEqvAndInf ChopAssoc EmptyOrChopEqv Prop04 Prop06 RightChopEqvChop*)
have *5*: $\vdash$ (*womega f*) =
(*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$
(*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)));*womega* (*f* $\wedge$ *finite*))
by (*metis 0 1 2 3 int-eq*)
have *6*: $\vdash$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$
(*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)));*womega* (*f* $\wedge$ *finite*)) =
((*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\vee$
(*womega* (*f* $\wedge$ *finite*)) $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))
using *4*
by (*metis 5 int-eq*)
have *7*: $\vdash$ ((*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\vee$
(*womega* (*f* $\wedge$ *finite*)) $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*))) =
((*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\vee$
(*womega* (*f* $\wedge$ *finite*)))
by *auto*
show *?thesis* **using** *5 6 7*

by (*metis int-eq*)
qed

lemma *womega-imp-inf*:
$\vdash$ (*womega f*) $\longrightarrow$ *inf*
proof −
have *1*: $\vdash$ (*womega f*) = (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$ (*womega* (*f* $\wedge$ *finite*)))
using *womega-split-finite-inf* **by** *blast*
have *2*: $\vdash$ (*womega* (*f* $\wedge$ *finite*)) $\longrightarrow$ *inf*
by (*metis omega-d-def omega-imp-inf*)
have *3*: $\vdash$ *wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\longrightarrow$ *inf*
by (*meson AndChopB ChopAndB FiniteChopInfEqvInf Prop11 WPowerstar-inductL-var-equiv lift-imp-trans*)
show *?thesis* **using** *1 2 3* **by** *fastforce*
qed

lemma *WOmega-Omega-skip*:
$\vdash$ *womega skip* = *omega skip*
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite Omega-Wagner-1 Prop10 SkipChopFiniteImpFinite WOmega-Wagner-1 WPowerstar-skip-finite inteq-reflection omega-d-def omega-split-empty-schopstar-skip-finite womega-split-empty-more*)

lemma *WOmega-Omega*:
$\vdash$ (*womega f*) = ((*schopstar f*);((*f* $\wedge$ *inf*)) $\vee$ (*omega f*))
by (*metis SChopstar-WPowerstar inteq-reflection omega-d-def womega-split-finite-inf*)

lemma *womega-and-more-inf*:
$\vdash$ (*womega* (*f* $\wedge$ *more*)) $\longrightarrow$ *inf*
by (*simp add*: *womega-imp-inf*)

lemma *womega-fmore-inf-eq*:
$\vdash$ *womega fmore* = *inf*
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite Prop11 WOmega-Omega-skip WOmega-Wagner-1 WPowerstar-more-absorb inf-imp-omega-skip inteq-reflection womega-fmore-inf wpowerstar-skip-finite wpowerstar-skip-fmore*)

lemma *womega-skip-inf-eq*:
$\vdash$ *womega skip* = *inf*
by (*metis Prop11 WOmega-Omega-skip inf-imp-omega-skip inteq-reflection womega-skip-inf*)

lemma *womega-more-inf*:
$\vdash$ *womega more* $\longrightarrow$ *inf*
by (*simp add*: *womega-imp-inf*)

lemma *womega-more-inf-eq*:

$\vdash$ *womega more = inf*
by (*metis OrFiniteInf WOmega-subdist fmore-d-def int-eq int-iffI womega-fmore-inf-eq womega-more-inf*)

lemma *womega-true*:
$\vdash$ *womega #True = inf*
by (*metis AndInfEqvChopFalse MoreAndInfEqvInf Prop11 RightChopImpChop WOmegaWeakCoinduct int-simps*(*11*) *int-simps*(*17*) *inteq-reflection womega-imp-inf*)

lemma *OmegaChopOmega*:
$\vdash$ (*omega f*);*g* = (*omega f*)
by (*metis AndInfChopEqvAndInf Prop10 inteq-reflection omega-imp-inf*)

lemma *WOmegaChopWOmega*:
$\vdash$ (*womega f*);*g* = (*womega f*)
by (*metis ChopSChopdef Prop04 Prop05 Prop12 WOmega-1 WPowerstar-tc int-iffD2 int-iffI lift-imp-trans womega-imp-inf*)

lemma *WOmegYieldsEvvNotWOmega*:
$\vdash$ (*womega f*) *yields g* = (¬(*womega f*))
by (*simp add*: *WOmegaChopWOmega WOmega-1 int-iffD2 int-iffI yields-d-def*)

3.6.3 Properties of Omega

lemma *NotOmegaInf*:
$\vdash$ ¬(*omega* (*inf*))
proof −
have *1*: $\vdash$ *omega* (*inf*) = ((*inf* ∧ *more*) ∧ *finite*);*omega inf*
by (*metis OmegaUnroll schop-d-def*)
have *2*: $\vdash$ ((*inf* ∧ *more*) ∧ *finite*) = *#False*
unfolding *finite-d-def* **by** *auto*
have *3*: $\vdash$ *#False*;*omega inf* = *#False*
by (*metis AndInfChopEqvAndInf int-eq int-simps*(*22*))
from *1 2 3* **show** *?thesis*
by (*metis TrueW int-simps*(*3*) *inteq-reflection*)
qed

lemma *InfImpOmegaLenPlusOne*:
$\vdash$ *inf* ⟶ *omega* (*len*(*Suc n*))
proof −
have *1*: $\vdash$ *inf* ⟶ ((*len Suc n* ∧ *more*) ∧ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs len-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis*
using *OmegaWeakCoinduct*[*of LIFT inf LIFT len* (*Suc n*)] *1*
by (*metis schop-d-def*)
qed

lemma *OmegaLenPlusOneEqvInf*:
$\vdash$ *omega* (*len*(*Suc n*)) = *inf*
by (*simp add*: *InfImpOmegaLenPlusOne int-iffI omega-imp-inf*)

lemma *OmegaSkipEqvInf*:
$\vdash$ *omega skip* = *inf*
proof −
have *1*: $\vdash$ *skip* = (*len 1*)
by (*simp add*: *len-d-def wpower-d-def*)
(*metis ChopEmpty inteq-reflection*)
have *2*: $\vdash$ *omega skip* = *omega* (*len 1*)
using *1* **by** (*metis OmegaUnroll inteq-reflection*)
from *2* **show** *?thesis* **using** *OmegaLenPlusOneEqvInf* **by** *fastforce*
qed

lemma *InfImpOmegaTrue*:
$\vdash$ *inf* $\longrightarrow$ *omega* #*True*
proof −
have *1*: $\vdash$ *inf* $\longrightarrow$ ((#*True* $\wedge$ *more*) $\wedge$ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis*
using *OmegaWeakCoinduct*[*of LIFT inf LIFT* #*True*] *1*
by (*metis schop-d-def*)
qed

lemma *OmegaTrueEqvInf*:
$\vdash$ *omega* #*True* = *inf*
by (*simp add*: *InfImpOmegaTrue int-iffI omega-imp-inf*)

lemma *InfImpOmegaMore*:
$\vdash$ *inf* $\longrightarrow$ *omega more*
using *OmegaWeakCoinduct*[*of LIFT inf LIFT more*]
proof −
have *1*: $\vdash$ *inf* $\longrightarrow$ ((*more* $\wedge$ *more*) $\wedge$ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis* **using** *OmegaWeakCoinduct*[*of LIFT inf LIFT more*] *1* **by** (*metis schop-d-def*)
qed

lemma *OmegaMoreEqvInf*:
$\vdash$ *omega more* = *inf*
by (*simp add*: *InfImpOmegaMore int-iffI omega-imp-inf*)

lemma *InfImpOmegaFinite*:
$\vdash$ *inf* $\longrightarrow$ *omega finite*
proof −
have *1*: $\vdash$ *inf* $\longrightarrow$ ((*finite* $\wedge$ *more*) $\wedge$ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis* **using** *OmegaWeakCoinduct*[*of LIFT inf LIFT finite*] *1* **by** (*metis schop-d-def*)
qed

lemma *OmegaFiniteEqvInf*:

$\vdash$ *omega finite = inf*
by (*simp add*: *InfImpOmegaFinite int-iffI omega-imp-inf*)

lemma *BoxStateAndInfImpWOmegaBoxState*:
$\vdash$ $\Box$(*init w*) $\wedge$ *inf* $\longrightarrow$ *womega* ($\Box$(*init w*))
using *WOmegaWeakCoinduct*[*of LIFT* ($\Box$(*init w*)) *LIFT* $\Box$(*init w*)]
by (*metis BoxStateChopBoxAndInfImpBox BoxStateChopBoxEqvBox WOmega-iso inteq-reflection womega-and-inf*)

lemma *BoxStateAndInfImpOmegaBoxState*:
$\vdash$ $\Box$(*init w*) $\wedge$ *inf* $\longrightarrow$ *omega* ($\Box$(*init w*))
proof −
have *1*: $\vdash$ (($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*);($\Box$ (*init w*) $\wedge$ *inf*) =
($\Box$ (*init w*) $\wedge$ (*more* $\wedge$ *finite*);*inf*)
using *BoxStateAndChopEqvChop*[*of w LIFT* (*more* $\wedge$ *finite*) *LIFT inf*]
by (*metis AndMoreAndFiniteEqvAndFmore fmore-d-def int-eq*)
have *2*: $\vdash$ (*more* $\wedge$ *finite*);*inf* = *inf*
by (*metis AndInfEqvChopFalse ChopAssoc FiniteChopMoreEqvMore FmoreEqvSkipChopFinite MoreAndInfEqvInf MoreEqvSkipChopTrue fmore-d-def infinite-d-def inteq-reflection*)
have *3*: $\vdash$ $\Box$ (*init w*) $\wedge$ *inf* $\longrightarrow$ (($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*);($\Box$ (*init w*) $\wedge$ *inf*)
by (*metis 1 2 Prop11 int-eq*)
show *?thesis* **using** *OmegaWeakCoinduct*[*of LIFT* ($\Box$(*init w*) $\wedge$ *inf*) *LIFT* $\Box$(*init w*)] *3*
by (*metis schop-d-def*)
qed

lemma *WOmegaBoxStateImpBoxState*:
$\vdash$ *womega* ($\Box$(*init w*)) $\longrightarrow$ $\Box$(*init w*)
proof −
have *1*: $\vdash$ *womega* ($\Box$(*init w*)) $\longrightarrow$ *init w*
by (*meson BoxElim Prop11 StateChopExportA WOmegaUnroll WOmega-iso lift-imp-trans*)
have *2*: $\vdash$ *womega* ($\Box$(*init w*)) $\longrightarrow$ ($\Box$(*init w*) $\wedge$ *more*); *womega* ($\Box$(*init w*))
by (*simp add*: *WOmegaUnroll int-iffD1*)
have *21*: $\vdash$ (($\Box$(*init w*) $\wedge$ *more*)) $\longrightarrow$ $\bigcirc$($\Box$(*init w*))
by (*metis BoxImpNowAndWeakNext Prop01 Prop12 WnextEqvEmptyOrNext empty-d-def inteq-reflection lift-and-com*)
have *22*: $\vdash$ *womega* ($\Box$(*init w*)) $\longrightarrow$ *more*
by (*meson Prop11 WOmega-and-more-imp-more lift-imp-trans womega-split-empty-more*)
have *23*: $\vdash$ $\bigcirc$($\Box$(*init w*)) $\longrightarrow$ $\bigcirc$ (*wpowerstar* ($\Box$ (*init w*)))
by (*simp add*: *NextImpNext WPowerstar-ext*)
have *24*: $\vdash$ *womega* ($\Box$(*init w*)) $\longrightarrow$ ($\bigcirc$(*wpowerstar* ($\Box$ (*init w*)))); (*womega* ($\Box$(*init w*)))
by (*metis 21 23 LeftChopImpChop WOmegaUnroll int-eq lift-imp-trans*)
have *10*: $\vdash$ *womega* ($\Box$(*init w*)) $\longrightarrow$ $\bigcirc$ (*womega* ($\Box$(*init w*)))
using *WOmega-star-1*[*of LIFT* ($\Box$(*init w*))]
by (*metis 24 NextChop inteq-reflection*)
show *?thesis* **using** *BoxIntro*[*of LIFT womega* ($\Box$(*init w*)) *LIFT* (*init w*)]
by (*simp add*: *1 10 Prop01 Prop05 Prop12 womega-imp-inf*)
qed

lemma *OmegaBoxStateImpBoxState*:
$\vdash$ *omega* ($\Box$(*init w*)) $\longrightarrow$ $\Box$(*init w*)
by (*meson Prop03 WOmegaBoxStateImpBoxState WOmega-Omega lift-imp-trans*)

lemma *OmegaIntro*:
assumes $\vdash$ $h \longrightarrow (f \wedge fmore);h$
shows $\vdash$ $h \longrightarrow$ *omega f*
using *assms*
by (*metis AndMoreSChopEqvAndFmoreChop OmegaWeakCoinduct inteq-reflection*)

lemma *WOmegaEqvRule*:
assumes $\vdash$ $f = g$
shows $\vdash$ *womega f* = *womega g*
using *assms* **using** *int-eq* **by** *force*

lemma *OmegaEqvRule*:
assumes $\vdash$ $f = g$
shows $\vdash$ *omega f* = *omega g*
using *assms* **using** *int-eq* **by** *force*

lemma *AndWOmegaA*:
$\vdash$ *womega* ($f \wedge g$) $\longrightarrow$ *womega f*
by (*meson WOmega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaA*:
$\vdash$ *omega* ($f \wedge g$) $\longrightarrow$ *omega f*
by (*meson Omega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndWOmegaB*:
$\vdash$ *womega* ($f \wedge g$) $\longrightarrow$ *womega g*
by (*meson WOmega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaB*:
$\vdash$ *omega* ($f \wedge g$) $\longrightarrow$ *omega g*
by (*meson Omega-iso Prop12 int-iffD2 lift-and-com*)

lemma *BaWOmegaImpWOmega*:
$\vdash$ *ba* ($f \longrightarrow g$) $\longrightarrow$ *womega f* $\longrightarrow$ *womega g*
proof −
have *1*: $\vdash$ *ba* ($f \longrightarrow g$) $\wedge$ ($f \wedge more$);*womega f* $\longrightarrow$(($f \longrightarrow g$) $\wedge$ ($f \wedge more$));(($f \longrightarrow g$) $\wedge$ *womega f*)
using *BaAndChopImport*[*of LIFT* ($f \longrightarrow g$) *LIFT* ($f \wedge more$) *LIFT womega f*]
by *blast*
have *2*: $\vdash$ ($f \longrightarrow g$) $\wedge$ ($f \wedge more$) $\longrightarrow$ ($g \wedge more$)
by *auto*
have *3*: $\vdash$ ($f \longrightarrow g$) $\wedge$ *womega f* $\longrightarrow$ *womega f*
by *auto*

have *4*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *more*);*womega f* $\longrightarrow$ (*g* $\wedge$ *more*); *womega f*
using *1 2 3*
by (*metis* (*no-types*, *lifting*) *AndChopB ChopAndB Prop10 int-eq lift-imp-trans*)
have *5*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *more*);*womega f* $\longrightarrow$ ((*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *more*));(*ba*(*f* $\longrightarrow$ *g*) $\wedge$ *womega f*)
using *BaAndChopImportB* **by** *blast*
have *6*: $\vdash$ ((*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *more*));(*ba*(*f* $\longrightarrow$ *g*) $\wedge$ *womega f*) $\longrightarrow$
(*g* $\wedge$ *more*);(*ba*(*f* $\longrightarrow$ *g*) $\wedge$ *womega f*)
using *2 LeftChopImpChop* **by** *blast*
have *61*: $\vdash$ *womega f* = (*f* $\wedge$ *more*);*womega f*
by (*simp add*: *WOmegaUnroll*)
have *62*: $\vdash$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *womega f*) $\longrightarrow$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *more*);*womega f*)
by (*metis 61 ChopEmpty Prop11 int-eq*)
have *7*: $\vdash$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *womega f*) $\longrightarrow$(*g* $\wedge$ *more*);(*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *womega f*)
using *62 5 6*
by (*meson lift-imp-trans*)
have *8*: $\vdash$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *womega f*) $\longrightarrow$ *womega g*
using *7*
using *WOmegaWeakCoinduct* **by** *blast*
from *8* **show** *?thesis* **by** *fastforce*
qed

lemma *BaOmegaImpOmega*:
$\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *omega f* $\longrightarrow$ *omega g*
proof −
have *1*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*);*omega f* $\longrightarrow$((*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*));((*f* $\longrightarrow$ *g*) $\wedge$ *omega f*)
using *BaAndChopImport*[*of LIFT* (*f* $\longrightarrow$ *g*) *LIFT* (*f* $\wedge$ *fmore*) *LIFT omega f*]
by *blast*
have *2*: $\vdash$ (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*) $\longrightarrow$ (*g* $\wedge$ *fmore*)
by *auto*
have *3*: $\vdash$ (*f* $\longrightarrow$ *g*) $\wedge$ *omega f* $\longrightarrow$ *omega f*
by *auto*
have *4*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*);*omega f* $\longrightarrow$ (*g* $\wedge$ *fmore*); *omega f*
using *1 2 3*
by (*metis* (*no-types*, *lifting*) *AndChopB ChopAndB Prop10 int-eq lift-imp-trans*)
have *5*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*);*omega f* $\longrightarrow$ ((*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*));(*ba*(*f* $\longrightarrow$ *g*) $\wedge$ *omega f*)
using *BaAndChopImportB* **by** *blast*
have *6*: $\vdash$ ((*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*));(*ba*(*f* $\longrightarrow$ *g*) $\wedge$ *omega f*) $\longrightarrow$
(*g* $\wedge$ *fmore*);(*ba*(*f* $\longrightarrow$ *g*) $\wedge$ *omega f*)
using *2 LeftChopImpChop* **by** *blast*
have *61*: $\vdash$ *omega f* = (*f* $\wedge$ *fmore*);*omega f*
by (*metis AndMoreAndFiniteEqvAndFmore OmegaUnroll inteq-reflection schop-d-def*)
have *62*: $\vdash$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *omega f*) $\longrightarrow$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *fmore*);*omega f*)
by (*metis 61 ChopEmpty Prop11 int-eq*)
have *7*: $\vdash$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *omega f*) $\longrightarrow$(*g* $\wedge$ *fmore*);(*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *omega f*)
using *62 5 6*
by (*meson lift-imp-trans*)
have *8*: $\vdash$ (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *omega f*) $\longrightarrow$ *omega g*
using *7 OmegaIntro* **by** *blast*
from *8* **show** *?thesis* **by** *fastforce*

qed

lemma *BaWOmegaEqvWOmega*:
$\vdash$ *ba* (*f* = *g*) $\longrightarrow$ (*womega f* = *womega g*)
proof −
have *0*: $\vdash$ (*f* = *g*) = ((*f* $\longrightarrow$ *g*) $\wedge$ (*g* $\longrightarrow$ *f*)) **by** *fastforce*
have *1*: $\vdash$ *ba* (*f* = *g*) = (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *ba* (*g* $\longrightarrow$ *f*)) **by** (*metis 0 BaAndEqv int-eq*)
have *2*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*womega f* $\longrightarrow$ *womega g*) **using** *BaWOmegaImpWOmega* **by** *blast*
have *3*: $\vdash$ *ba* (*g* $\longrightarrow$ *f*) $\longrightarrow$ (*womega g* $\longrightarrow$ *womega f*) **using** *BaWOmegaImpWOmega* **by** *blast*
have *4*: $\vdash$ *ba* (*f* = *g*) $\longrightarrow$ (*womega f* $\longrightarrow$ *womega g*) $\wedge$ (*womega g* $\longrightarrow$ *womega f*) **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash$ ((*womega f* $\longrightarrow$ *womega g*) $\wedge$ (*womega g* $\longrightarrow$ *womega f*)) = (*womega f* = *womega g*) **by** *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaOmegaEqvOmega*:
$\vdash$ *ba* (*f* = *g*) $\longrightarrow$ (*omega f* = *omega g*)
proof −
have *0*: $\vdash$ (*f* = *g*) = ((*f* $\longrightarrow$ *g*) $\wedge$ (*g* $\longrightarrow$ *f*)) **by** *fastforce*
have *1*: $\vdash$ *ba* (*f* = *g*) = (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *ba* (*g* $\longrightarrow$ *f*)) **by** (*metis 0 BaAndEqv int-eq*)
have *2*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*omega f* $\longrightarrow$ *omega g*) **using** *BaOmegaImpOmega* **by** *blast*
have *3*: $\vdash$ *ba* (*g* $\longrightarrow$ *f*) $\longrightarrow$ (*omega g* $\longrightarrow$ *omega f*) **using** *BaOmegaImpOmega* **by** *blast*
have *4*: $\vdash$ *ba* (*f* = *g*) $\longrightarrow$ (*omega f* $\longrightarrow$ *omega g*) $\wedge$ (*omega g* $\longrightarrow$ *omega f*) **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash$ ((*omega f* $\longrightarrow$ *omega g*) $\wedge$ (*omega g* $\longrightarrow$ *omega f*)) = (*omega f* = *omega g*) **by** *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaAndWOmegaImport*:
$\vdash$ *ba f* $\wedge$ *womega g* $\longrightarrow$ *womega* (*f* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ (*g* $\longrightarrow$ (*f* $\wedge$ *g*)) **by** *auto*
hence *2*: $\vdash$ *ba f* $\longrightarrow$ *ba* (*g* $\longrightarrow$ *f* $\wedge$ *g*) **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba* (*g* $\longrightarrow$ *f* $\wedge$ *g*) $\longrightarrow$ *womega g* $\longrightarrow$ *womega* (*f* $\wedge$ *g*) **by** (*rule BaWOmegaImpWOmega*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndOmegaImport*:
$\vdash$ *ba f* $\wedge$ *omega g* $\longrightarrow$ *omega* (*f* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ (*g* $\longrightarrow$ (*f* $\wedge$ *g*)) **by** *auto*
hence *2*: $\vdash$ *ba f* $\longrightarrow$ *ba* (*g* $\longrightarrow$ *f* $\wedge$ *g*) **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba* (*g* $\longrightarrow$ *f* $\wedge$ *g*) $\longrightarrow$ *omega g* $\longrightarrow$ *omega* (*f* $\wedge$ *g*) **by** (*rule BaOmegaImpOmega*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *InfAndChop*:
$\vdash$ (*inf* $\wedge$ (*f* $\wedge$ *fmore*);*g*) = (*f* $\wedge$ *fmore*);(*g* $\wedge$ *inf*)
by (*meson ChopAndInf Prop04 lift-and-com*)

lemma *InfImportBox*:

$\vdash (inf \wedge \Box\ f) = \Box\ (inf \wedge f)$
by (*metis BoxAndBoxEqvBoxRule FiniteChopEqvDiamond FiniteChopFiniteEqvFinite InfEqvNotFinite NotDiamondNotEqvBox OrFiniteInf finite-d-def int-eq*)

lemma *InfImportBoxImp*:
$\vdash (inf \wedge \Box\ (f \longrightarrow g)\ ) = \Box\ (\ (inf \longrightarrow f) \longrightarrow (inf \wedge g))$
proof −
have *1*: $\vdash (inf \wedge (f \longrightarrow g)) = ((inf \longrightarrow f) \longrightarrow (inf \wedge g))$
by *auto*
show *?thesis*
by (*metis 1 InfImportBox inteq-reflection*)
qed

lemma *OmegaImpAOmega*:
$\vdash$ *omega f* $\longrightarrow$ *aomega f*
using *AOmegaWeakCoInduct*[*of LIFT omega f f*]
by (*metis OmegaUnroll Prop10 Prop11 inteq-reflection lift-and-com omega-imp-inf schop-d-def*)

lemma *AOmegaImpOmega*:
$\vdash$ *aomega f* $\longrightarrow$ *omega f*
using *OmegaWeakCoinduct*[*of LIFT aomega f f*]
by (*simp add*: *AOmegaUnroll int-iffD1 schop-d-def*)

lemma *OmegaEqvAOmega*:
$\vdash$ *omega f* = *aomega f*
using *OmegaImpAOmega AOmegaImpOmega*
by (*metis int-iffI*)

lemma *WOmega-Finite-Import*:
$\vdash \neg$((*womega f*) $\wedge$ *finite*)
by (*metis MP NotEmptyAndInf Prop05 Prop13 finite-d-def int-eq int-simps*(*32*) *womega-imp-inf*)

lemma *WOmega-Inf-Import*:
$\vdash$ ((*womega f*) $\wedge$ *inf*) = ((*f*;(*wpowerstar f*) $\wedge$ *inf*) $\vee$ ((*womega* (*f* $\wedge$ *finite*)) $\wedge$ *inf*))
proof −
have *1*: $\vdash$ (*womega f*) = (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$ (*womega* (*f* $\wedge$ *finite*)))
using *womega-split-finite-inf* **by** *blast*
have *2*: $\vdash$ (*f*;(*wpowerstar f*) $\wedge$ *inf*) = ((*wpowerstar f*) $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*metis WPowerstar-And-Inf-Eqv-WPowerplus-And-Inf WPowerstar-Inf-Import WPowerstar-slide-var inteq-reflection*)

have *4*: $\vdash$ ((*womega f*) $\wedge$ *inf*) =
((*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\wedge$ *inf*) $\vee$ ((*womega* (*f* $\wedge$ *finite*)) $\wedge$ *inf*))
using *1* **by** *fastforce*
have *5*: $\vdash$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\wedge$ *inf*) = *wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*metis ChopAndInf Prop10 inteq-reflection womega-and-inf womega-imp-inf*)
show *?thesis*
by (*metis 2 4 5 WPowerstar-import-finite inteq-reflection*)

qed

lemma *WOmega-And-Finite-Inf-Import*:
$\vdash$ $(womega\ (f \wedge finite) \wedge inf) = (womega\ ((f \wedge finite)))$
by (*meson Prop10 Prop11 womega-imp-inf*)

lemma *WOmega-Inf-Import-var-1*:
$\vdash ((womega\ f) \wedge inf) = (wpowerstar\ (f \wedge finite);((f \wedge inf)) \vee womega\ ((f \wedge finite)\ ))$
by (*meson Prop04 Prop10 womega-imp-inf womega-split-finite-inf*)

lemma *WOmega-Inf-Import-var-2*:
$\vdash ((womega\ (f)) \wedge inf) = (womega\ (f))$
by (*meson Prop10 Prop11 womega-imp-inf*)

lemma *womega-4*:
$\vdash (womega\ (\ (g \vee (womega\ (f)))\ )) =$
$(\ (womega\ (g\ )) \vee wpowerstar\ g;(womega\ (f))\ )$
using *WOmega-or-unfold*[*of g LIFT* $(womega\ (f))$]
by (*metis* (*no-types*, *lifting*) *ChopAssoc Prop10 WOmegaChopWOmega WOmega-and-more-imp-more inteq-reflection womega-split-empty-more*)

lemma *womega-5*:
$\vdash (womega\ (\ (g \vee h;(womega\ (f\ )))\ )) =$
$(\ (womega\ (g\ )) \vee wpowerstar\ g;(h;(womega\ (f)))\ )$
proof −
have *1*: $\vdash (g^{\mathcal{W}} \vee$
$(wpowerstar\ g;(h;f^{\mathcal{W}} \wedge more));(g \vee h;f^{\mathcal{W}})^{\mathcal{W}}) =$
$(g \vee h;f^{\mathcal{W}})^{\mathcal{W}}$
using *WOmega-or-unfold*[*of g LIFT* $(h;(womega\ (f\ )))$] **by** *blast*
have *2*: $\vdash (wpowerstar\ g;(h;f^{\mathcal{W}} \wedge more));(g \vee h;f^{\mathcal{W}})^{\mathcal{W}} = (wpowerstar\ g);((h;f^{\mathcal{W}} \wedge more);((g \vee h;f^{\mathcal{W}})^{\mathcal{W}}))$
by (*meson ChopAssoc Prop11*)
have *3*: $\vdash\ ((h;f^{\mathcal{W}} \wedge more)) = (h;f^{\mathcal{W}})$
by (*metis MoreAndInfEqvInf Prop10 Prop12 RightChopImpMoreRule inteq-reflection womega-imp-inf*)
have *4*: $\vdash (h;f^{\mathcal{W}});((g \vee h;f^{\mathcal{W}})^{\mathcal{W}}) =\ (h;f^{\mathcal{W}})$
by (*metis ChopAssoc WOmegaChopWOmega inteq-reflection*)
show *?thesis*
by (*metis 1 2 3 4 inteq-reflection*)
qed

lemma *womega-6*:
$\vdash womega\ ((womega\ (f))\ ) = womega\ (f\ )$
by (*meson Prop04 WOmegaEqvRule WOmega-Inf-Import-var-2 womega-and-inf*)

lemma *womega-7*:
$\vdash (womega\ ((g;(womega\ (f)))\ )) = g;(womega\ (f\ ))$
proof −
have *1*: $\vdash (womega\ ((g;(womega\ (f)))\ )) = (g;(womega\ (f))) \ ;\ (womega\ ((g;(womega\ (f)))\ ))$

by (*meson AndChopA LeftChopImpChop Prop11 WOmegaUnroll WOmega-star-1 WPowerstar-ext lift-imp-trans*)
have *2*: $\vdash$ (*g*;(*womega* (*f*))) ; (*womega* ((*g*;(*womega* (*f*))))) = (*g*;(*womega* (*f*)))
by (*metis ChopAssoc WOmegaChopWOmega inteq-reflection*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *womega-8*:
$\vdash$ (*womega* (*f*));(*wpowerstar* (*womega* (*f*))) = *womega* (*f*)
using *WOmega-star-5* **by** *auto*

lemma *womega-9*:
$\vdash$ (*g*;(*womega* (*f*)));(*wpowerstar* (*g*;(*womega* (*f*)))) = *g*;*womega* (*f*)
by (*metis inteq-reflection womega-7 womega-8*)

lemma *womega-10*:
$\vdash$ ((*g* $\vee$ (*womega* (*f*))));(*wpowerstar* ((*g* $\vee$ (*womega* (*f*))))) =
(*g*;*wpowerstar g* $\vee$ *wpowerstar g*;(*womega* (*f*)))
proof −
have *1*: $\vdash$ (*wpowerstar* ((*g* $\vee$ (*womega* (*f*))))) =
wpowerstar (*wpowerstar g*;*wpowerstar* ((*f*)$^{\mathcal{W}}$))
using *WPowerstar-denest*[*of g LIFT* (*womega* (*f*))] **by** *auto*
have *2*: $\vdash$ *wpowerstar* ((*f*)$^{\mathcal{W}}$) = (*empty* $\vee$ ((*f*)$^{\mathcal{W}}$))
by (*simp add*: *WOmega-star-4*)
have *3*: $\vdash$ (*wpowerstar g*;(*empty* $\vee$ ((*f*)$^{\mathcal{W}}$))) =
(*wpowerstar g* $\vee$ (*wpowerstar g*);((*f*)$^{\mathcal{W}}$))
by (*metis ChopEmpty ChopOrEqv inteq-reflection*)
have *4*: $\vdash$ *wpowerstar* (*wpowerstar g* $\vee$ (*wpowerstar g*);((*f*)$^{\mathcal{W}}$)) =
wpowerstar (*wpowerstar g*;*wpowerstar* ((*wpowerstar g*);((*f*)$^{\mathcal{W}}$)))
using *WPowerstar-denest* **by** (*metis WPowerstar-invol inteq-reflection*)
have *5*: $\vdash$ *wpowerstar* (*wpowerstar g*;*wpowerstar* ((*wpowerstar g*);((*f*)$^{\mathcal{W}}$))) =
wpowerstar g;*wpowerstar*((*wpowerstar g*);((*f*)$^{\mathcal{W}}$))
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc WOmegaChopWOmega WPowerstar-denest WPowerstar-denest-var inteq-reflection*)
have *6*: $\vdash$ *wpowerstar*((*wpowerstar g*);((*f*)$^{\mathcal{W}}$)) =
(*empty* $\vee$ *wpowerstar g*;(*wpowerstar* ((*f*)$^{\mathcal{W}}$;*wpowerstar g*);(*f*)$^{\mathcal{W}}$))
using *WPowerstar-star-prod-unfold*[*of LIFT* (*wpowerstar g*) *LIFT* ((*f*)$^{\mathcal{W}}$)]
by *auto*
have *7*: $\vdash$ (*wpowerstar g*;(*wpowerstar* ((*f*)$^{\mathcal{W}}$;*wpowerstar g*);(*f*)$^{\mathcal{W}}$)) =
(*wpowerstar g*;((*empty* $\vee$ ((*f*)$^{\mathcal{W}}$));((*f*)$^{\mathcal{W}}$)))
by (*metis WOmegaChopWOmega WOmega-star-4 int-simps*(*1*) *inteq-reflection*)
have *8*: $\vdash$ (*wpowerstar g*;((*empty* $\vee$ ((*f*)$^{\mathcal{W}}$));((*f*)$^{\mathcal{W}}$))) =
(*wpowerstar g*;((*f*)$^{\mathcal{W}}$))
by (*metis 2 RightChopEqvChop WOmegaChopWOmega WPowerstar-induct-lvar-eq2 inteq-reflection*)
have *9*: $\vdash$ *wpowerstar*((*wpowerstar g*);((*f*)$^{\mathcal{W}}$)) = (*empty* $\vee$ (*wpowerstar g*;((*f*)$^{\mathcal{W}}$)))
by (*meson Prop06 WPowerstarEqv womega-9*)
have *10*: $\vdash$ (*wpowerstar* ((*g* $\vee$ (*womega* (*f*))))) =
wpowerstar g;(*empty* $\vee$ (*wpowerstar g*;((*f*)$^{\mathcal{W}}$)))
by (*metis 1 2 3 4 5 6 7 8 inteq-reflection*)
have *11*: $\vdash$ *wpowerstar g*;(*empty* $\vee$ (*wpowerstar g*;((*f*)$^{\mathcal{W}}$))) =

$(wpowerstar\ g \vee (wpowerstar\ g;((f\ )^{\mathcal{W}})\ ))$
by (*metis 10 2 3 WOmegaChopWOmega WPowerstar-denest-var inteq-reflection*)
have *12*: $\vdash (\ (g \vee (womega\ (f\ ))));\ (wpowerstar\ g \vee (wpowerstar\ g;((f\ )^{\mathcal{W}})\ )) =$
$(\ g;wpowerstar\ g \vee g;(wpowerstar\ g;((f\ )^{\mathcal{W}})\ ) \vee (womega\ (f\ )))$
by (*meson ChopOrEqv OrChopOr Prop04 Prop06 WOmegaChopWOmega int-simps*(*20*))
have *13*: $\vdash (g;(wpowerstar\ g;((f\ )^{\mathcal{W}})\ ) \vee (womega\ (f\ ))) =$
$(wpowerstar\ g;((f\ )^{\mathcal{W}})\ )$
by (*metis LeftChopEqvChop Prop04 WPowerstarChopEqvChopOrRule WPowerstar-tc WPowerstar-trans*)
have *14*: $\vdash (\ g;wpowerstar\ g \vee g;(wpowerstar\ g;((f\ )^{\mathcal{W}})\ ) \vee (womega\ (f))) =$
$(\ g;wpowerstar\ g \vee (wpowerstar\ g;((f\ )^{\mathcal{W}})\ ))$
using *13* **by** *auto*
show *?thesis* **by** (*metis 10 11 12 14 inteq-reflection*)
qed

lemma *womega-11*:
$\vdash (\ (g \vee h;(womega\ (f\ ))));(wpowerstar\ (\ (g \vee h;(womega\ (f\ )))))) =$
$(\ g;wpowerstar\ g \vee wpowerstar\ g;(h;(womega\ (f\ )))\ )$
proof −
have *1*: $\vdash (wpowerstar\ (\ (g \vee h;(womega\ (f\ ))))) =$
$wpowerstar\ (wpowerstar\ g;wpowerstar\ (h;(f\ )^{\mathcal{W}}))$
using *WPowerstar-denest*[*of g LIFT h;*(*womega* (*f*))] **by** *auto*
have *2*: $\vdash wpowerstar\ (h;(f\ )^{\mathcal{W}}) = (empty \vee h;((f\ )^{\mathcal{W}}))$
by (*meson Prop06 WPowerstarEqv womega-9*)
have *3*: $\vdash\ (wpowerstar\ g;(empty \vee h;((f\ )^{\mathcal{W}}))\ ) =$
$(wpowerstar\ g \vee (wpowerstar\ g);(h;(f\ )^{\mathcal{W}}))$
by (*metis ChopEmpty ChopOrEqv inteq-reflection*)
have *4*: $\vdash\ wpowerstar\ (wpowerstar\ g \vee (wpowerstar\ g);(h;(f\ )^{\mathcal{W}})) =$
$wpowerstar\ (wpowerstar\ g;wpowerstar\ ((wpowerstar\ g);(h;(f\ )^{\mathcal{W}})))$
using *WPowerstar-denest* **by** (*metis WPowerstar-invol inteq-reflection*)
have *5*: $\vdash wpowerstar\ (wpowerstar\ g;wpowerstar\ ((wpowerstar\ g);(h;(f\ )^{\mathcal{W}}))) =$
$wpowerstar\ g;wpowerstar((wpowerstar\ g);(h;(f\ )^{\mathcal{W}}))$
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc WOmegaChopWOmega WPowerstar-denest WPowerstar-denest-var inteq-reflection*)
have *6*: $\vdash wpowerstar((wpowerstar\ g);(h;(f\ )^{\mathcal{W}})) =$
$(empty \vee wpowerstar\ g;(wpowerstar\ ((h;(f\ )^{\mathcal{W}});wpowerstar\ g);(h;(f\ )^{\mathcal{W}})))$
using *WPowerstar-star-prod-unfold*[*of LIFT* (*wpowerstar g*) *LIFT* (*h;*(*f*)$^{\mathcal{W}}$)]
by *auto*
have *7*: $\vdash (\ wpowerstar\ g;(wpowerstar\ ((h;(f\ )^{\mathcal{W}});wpowerstar\ g);(h;(f\ )^{\mathcal{W}}))) =$
$(\ wpowerstar\ g;((empty \vee (h;(f\ )^{\mathcal{W}}));(h;(f\ )^{\mathcal{W}})))$
by (*metis 2 ChopAssoc WOmegaChopWOmega inteq-reflection*)
have *8*: $\vdash (\ wpowerstar\ g;((empty \vee (h;(f\ )^{\mathcal{W}}));(h;(f\ )^{\mathcal{W}}))) =$
$(\ \ wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ )$
by (*metis 2 RightChopEqvChop WPowerstar-slide-var inteq-reflection womega-9*)
have *9*: $\vdash\ wpowerstar((wpowerstar\ g);(h;(f\ )^{\mathcal{W}})) =$
$(empty \vee (\ \ wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ ))$
by (*meson 6 7 8 Prop06*)
have *10*: $\vdash (wpowerstar\ (\ (g \vee h;(womega\ (f\ )))))) =$
$wpowerstar\ g;(empty \vee (\ \ wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ ))$
by (*metis 1 2 3 4 5 6 7 8 inteq-reflection*)
have *11*: $\vdash\ wpowerstar\ g;(empty \vee (\ \ wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ )) =$

$(wpowerstar\ g \vee (wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ ))$
by (*metis* (*no-types*, *lifting*) *10 3 WOmegaChopWOmega WPowerstarEqv WPowerstar-denest-var inteq-reflection womega-7*)
have *12*: $\vdash (\ (g \vee h;(womega\ (f\ ))));\ (wpowerstar\ g \vee (wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ )) =$
$(\ g;wpowerstar\ g \vee g;(wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ ) \vee h;(womega\ (f\ )))$
by (*metis* (*no-types*, *lifting*) *10 11 WPowerstarChopEqvChopOrRule inteq-reflection womega-10 womega-7*)
have *13*: $\vdash (g;(wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ ) \vee h;(womega\ (f\ ))) =$
$(wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ )$
by (*metis LeftChopEqvChop Prop04 WPowerstarChopEqvChopOrRule WPowerstar-tc WPowerstar-trans*)
have *14*: $\vdash (\ g;wpowerstar\ g \vee g;(wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ ) \vee h;(womega\ (f\ ))) =$
$(\ g;wpowerstar\ g \vee (wpowerstar\ g;(h;(f\ )^{\mathcal{W}})\ ))$
using *13* **by** *auto*
show *?thesis* **by** (*metis 10 11 12 14 inteq-reflection*)
qed

lemma *WPowerstar-imp-finite-or-finite-chop-and-inf*:
$\vdash wpowerstar\ f \longrightarrow finite \vee finite;(f \wedge inf)$
proof −
have *1*: $\vdash wpowerstar\ f \longrightarrow finite;(empty \vee (f \wedge inf))$
by (*meson LeftChopImpChop Prop11 SChopstar-finite WPowerstar-SChopstar lift-imp-trans*)
have *2*: $\vdash finite;(empty \vee (f \wedge inf)) = (finite \vee finite;(f \wedge inf))$
by (*metis ChopEmpty ChopOrEqv int-eq*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *Inf-and-always-imp-not-wpowerstar*:
$\vdash inf \wedge \Box\ (f \longrightarrow finite) \longrightarrow \neg(wpowerstar\ f)$
proof −
have *1*: $\vdash (\neg\ (f \longrightarrow \neg\ inf)) = (f \wedge inf)$
by *fastforce*
have *2*: $\vdash (\neg\ (\neg\ inf);(\neg\ (f \longrightarrow \neg\ inf))\ ) = (\neg((\neg\ inf);(f \wedge inf)))$
by (*metis 1 int-simps*(*15*) *inteq-reflection*)
show *?thesis*
using *WPowerstar-imp-finite-or-finite-chop-and-inf* **unfolding** *always-d-def sometimes-d-def finite-d-def*
using *2* **by** *fastforce*
qed

lemma *WOmega-And-empty*:
$\vdash \neg(\ womega\ f \wedge empty)$
proof −
have *1*: $\vdash (\ womega\ f \wedge empty) =$
$(\ (\ (schopstar\ f);(\ (f \wedge inf)) \vee (omega\ f)\ ) \wedge empty)$
using *WOmega-Omega* **by** (*metis int-simps*(*1*) *inteq-reflection*)
have *2*: $\vdash (\ (\ (schopstar\ f);(\ \ (f \wedge inf)) \vee (omega\ f)\ ) \wedge empty) =$
$(\ ((schopstar\ f);(\ \ (f \wedge inf)) \wedge empty) \vee$
$(omega\ f \wedge empty))$
by *fastforce*
have *3*: $\vdash \neg(omega\ f \wedge empty)$
using *NotEmptyAndInf omega-imp-inf* **by** *fastforce*

have *5*: $\vdash \neg((schopstar\ f);(f \wedge inf) \wedge empty)$
by (*metis* (*no-types*, *opaque-lifting*) *AndInfChopEqvAndInf ChopAndEmptyEqvEmptyChopEmpty ChopAndInf*
InitAndEmptyEqvAndEmpty NotEmptyAndInf StateAndEmptyChop inteq-reflection lift-and-com)
show *?thesis* **using** *1 2 3 5* **by** *fastforce*
qed

lemma *WOmega-And-More*:
$\vdash (\ womega\ f \wedge more) = (\ womega\ f\ )$
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite*
Prop10 WOmega-and-more-imp-more inteq-reflection lift-imp-trans womega-fmore-inf-eq womega-imp-inf)

end

3.7 Projection operator

theory *Projection*
imports *Omega*
begin

This theory introduces the projection operator fproj [10]. The projection operator is defined and we prove the soundness of the rules and axiom system. We also provide the infinite projection operator oproj which was defined in [13].

3.7.1 Definitions

primcorec *pfilt* :: $'a\ intervals \Rightarrow nat\ nellist \Rightarrow 'a\ intervals$
where
$pfilt\ \sigma\ ls = (case\ ls\ of\ (NNil\ n) \Rightarrow (NNil\ (nnth\ \sigma\ n)) \mid$
$(NCons\ n\ ls1) \Rightarrow (NCons\ (nnth\ \sigma\ n)\ (pfilt\ \sigma\ ls1)))$

simps-of-case *pfilt-code* [*code*, *simp*, *nitpick-simp*]: *pfilt.code*

definition *powerinterval* :: $('a::world)\ formula \Rightarrow 'a\ intervals \Rightarrow nat\ nellist \Rightarrow bool$
where $powerinterval\ F\ \sigma\ l =$
$(\forall\ (i::nat).\ (enat\ i) < nlength\ l \longrightarrow ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ ) \models F))$

definition *cppl* :: $('a::world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ intervals \Rightarrow nat\ nellist$
where $cppl\ f\ g\ \sigma = (\epsilon\ ls.\ \ nidx\ ls \wedge (nnth\ ls\ 0) = 0 \wedge powerinterval\ f\ \sigma\ ls \wedge$
$((nfinite\ ls \wedge nfinite\ \sigma \wedge\ (nlast\ ls) = the\text{-}enat(nlength\ \sigma))\ ) \vee$
$(\neg\ nfinite\ ls \wedge \neg nfinite\ \sigma\ ))\ \wedge$
$((pfilt\ \sigma\ ls) \models g))$

primcorec *lcppl* :: $('a::world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ intervals \Rightarrow nat\ nellist \Rightarrow nat\ intervals\ intervals$
where $lcppl\ \ f\ g\ \sigma\ ls =$

$(case\ ls\ of\ (NNil\ x) \Rightarrow (NNil\ (nmap\ (\lambda l.\ l{+}x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ x)))) \mid$
$(NCons\ x\ ls1) \Rightarrow$
$(case\ ls1\ of\ (NNil\ y) \Rightarrow (NNil\ (nmap\ (\lambda l.\ l{+}x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ y)))) \mid$
$(NCons\ y\ ls2) \Rightarrow (NCons\ (nmap\ (\lambda l.\ l{+}x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ y)))\ (lcppl\ f\ g\ \sigma\ ls1))))$

simps-of-case *lcppl-code* [*code*, *simp*, *nitpick-simp*]: *lcppl.code*

definition *fprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *fprojection-d F G* ≡ λσ. (∃ *ls*. *nidx ls* ∧ (*nnth ls 0*) = *0* ∧ *nfinite ls* ∧ *nfinite* σ ∧
(*nlast ls*) = *the-enat*(*nlength* σ) ∧
powerinterval F σ *ls* ∧ ((*pfilt* σ *ls*) ⊨ *G*)
)

definition *oprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *oprojection-d F G* ≡ λσ. (∃ *ls*. *nidx ls* ∧ (*nnth ls 0*) =*0* ∧ ¬*nfinite ls* ∧ ¬ *nfinite* σ ∧
powerinterval F σ *ls* ∧ ((*pfilt* σ *ls*) ⊨ *G*)
)

syntax
-fprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *fproj* -) [*84*,*84*] *83*)
-oprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *oproj* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-fprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *fproj* -) [*84*,*84*] *83*)
-oprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *oproj* -) [*84*,*84*] *83*)

translations
-fprojection-d ⇌ *CONST fprojection-d*
-oprojection-d ⇌ *CONST oprojection-d*

definition *ufprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *ufprojection-d F G* ≡ *LIFT*(¬(*F fproj* (¬ *G*)))

definition *uoprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *uoprojection-d F G* ≡ *LIFT*(¬(*F oproj* (¬ *G*)))

syntax
-ufprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *ufproj* -) [*84*,*84*] *83*)
syntax
-uoprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *uoproj* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-ufprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *ufproj* -) [*84*,*84*] *83*)
-uoprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *uoproj* -) [*84*,*84*] *83*)

translations

-ufprojection-d $\rightleftharpoons$ *CONST ufprojection-d*
-uoprojection-d $\rightleftharpoons$ *CONST uoprojection-d*

definition *fdp-d* :: (*'a*:: *world*) *formula* $\Rightarrow$ *'a formula*
where *fdp-d F* $\equiv$ *LIFT*(#*True fproj F*)

definition *fbp-d* :: (*'a*:: *world*) *formula* $\Rightarrow$ *'a formula*
where *fbp-d F* $\equiv$ *LIFT*(#*True ufproj F*)

definition *odp-d* :: (*'a*:: *world*) *formula* $\Rightarrow$ *'a formula*
where *odp-d F* $\equiv$ *LIFT*(#*True oproj F*)

definition *obp-d* :: (*'a*:: *world*) *formula* $\Rightarrow$ *'a formula*
where *obp-d F* $\equiv$ *LIFT*(#*True uoproj F*)

syntax
-fdp-d :: *lift* $\Rightarrow$ *lift* ((*fdp -*) [*88*] *87*)
-fbp-d :: *lift* $\Rightarrow$ *lift* ((*fbp -*) [*88*] *87*)
-odp-d :: *lift* $\Rightarrow$ *lift* ((*odp -*) [*88*] *87*)
-obp-d :: *lift* $\Rightarrow$ *lift* ((*obp -*) [*88*] *87*)

syntax (*ASCII*)
-fdp-d :: *lift* $\Rightarrow$ *lift* ((*fdp -*) [*88*] *87*)
-fbp-d :: *lift* $\Rightarrow$ *lift* ((*fbp -*) [*88*] *87*)
-odp-d :: *lift* $\Rightarrow$ *lift* ((*odp -*) [*88*] *87*)
-obp-d :: *lift* $\Rightarrow$ *lift* ((*obp -*) [*88*] *87*)

translations
-fdp-d $\rightleftharpoons$ *CONST fdp-d*
-fbp-d $\rightleftharpoons$ *CONST fbp-d*
-odp-d $\rightleftharpoons$ *CONST odp-d*
-obp-d $\rightleftharpoons$ *CONST obp-d*

3.7.2 Lemmas

pfilt Lemmas

lemma *pfilt-nmap*:
pfilt nell ls = *nmap* (λx. (*nnth nell x*)) *ls*
proof (*coinduction arbitrary*: *nell ls*)
case (*Eq-nellist nell1*)
then show *?case* **by** *simp* (*metis nellist.case-eq-if*)
qed

lemma *pfilt-nlength*:
nlength(*pfilt nell ls*) = *nlength ls*
by (*simp add*: *pfilt-nmap*)

lemma *pfilt-nnth*:

assumes $i \leq nlength\ (pfilt\ nell\ ls)$
shows $(nnth\ (pfilt\ nell\ ls)\ i) = (nnth\ nell\ (nnth\ ls\ i))$
using *assms*
by (*simp add*: *pfilt-nmap*)

lemma *pfilt-expand*:
$(nell1 = (pfilt\ nell\ ls)) =$
$(\ nlength\ nell1 = nlength\ ls \wedge$
$(\forall\ (i::nat).\ i \leq nlength\ nell1 \longrightarrow (nnth\ nell1\ i) = (nnth\ nell\ (nnth\ ls\ i))))$
by (*metis nellist-eq-nnth-eq pfilt-nlength pfilt-nnth*)

lemma *pfilt-nfuse*:
$pfilt\ nell\ (nfuse\ ls1\ ls2) = (nfuse\ (pfilt\ nell\ ls1)\ (pfilt\ nell\ ls2))$
using *pfilt-nmap*[*of nell* (*nfuse ls1 ls2*)] *pfilt-nmap*[*of nell ls1*] *pfilt-nmap*[*of nell ls2*]
by (*simp add*: *nmap-nfuse*)

lemma *nfuse-pfilt-nlength*:
shows $nlength\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls) =$
$nlength\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
$(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x.\ x -\ (nnth\ ls\ n))\ (ndropn\ n\ ls))))$
by (*metis add.right-neutral linorder-le-cases ndropn-all nfuse-nlength nfuse-ntaken-ndropn-nlength nlength-NNil nlength-nmap ntaken-all pfilt-nlength*)

lemma *nfuse-pfilt-nnth-a*:
assumes *nidx ls*
$(nnth\ ls\ 0) = 0$
$(nlast\ ls) = the\text{-}enat(nlength\ nell)$
nfinite nell
nfinite ls
$i \leq nlength\ ls$
$n \leq nlength\ ls$
shows $nnth\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls)\ i =$
$nnth\ nell\ (nnth\ ls\ i)$
using *assms*
by (*metis linorder-le-cases ndropn-all nfinite-conv-nlength-enat nfuse-ntaken-ndropn ntaken-all pfilt-nlength pfilt-nnth*)

lemma *nfuse-pfilt-nnth-a-alt*:
assumes *nidx ls*
$(nnth\ ls\ 0) = 0$
$\neg nfinite\ nell$
$\neg nfinite\ ls$
$i \leq nlength\ ls$
$n \leq nlength\ ls$
shows $nnth\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls)\ i =$
$nnth\ nell\ (nnth\ ls\ i)$
using *assms*
by (*metis linorder-le-cases nfinite-ntaken nfuse-ntaken-ndropn ntaken-all pfilt-nlength pfilt-nnth*)

lemma *nfuse-pfilt-nnth-b*:
assumes *nidx ls*
nnth ls 0 = 0
(nlast ls) = the-enat(nlength nell)
nfinite nell
nfinite ls
i ≤ nlength ls
n ≤ nlength ls
shows *nnth (nfuse (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))*
(pfilt (ndropn (nnth ls n) nell) (nmap (λx. x − (nnth ls n)) (ndropn n ls)))) i =
nnth nell (nnth ls i)
proof −
have *1*: *i ≤ nlength(nfuse (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))*
(pfilt (ndropn (nnth ls n) nell) (nmap (λx. x − (nnth ls n)) (ndropn n ls))))
using *assms*
by (*metis nfuse-pfilt-nlength pfilt-nlength*)
have *2*: *nlast(pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) = nnth nell (nnth ls n)*
using *assms pfilt-nnth*[*of - (ntaken (nnth ls n) nell) (ntaken n ls)*]
by (*simp add*: *nlength-eq-enat-nfiniteD nnth-nlast ntaken-nnth pfilt-nlength*)
have *3*: *nfirst(pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls))) =*
nnth (pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls))) 0
by (*metis ndropn-0 ndropn-nfirst*)
have *4*: *nnth (pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls))) 0 =*
nnth (ndropn (nnth ls n) nell) (nnth (nmap (λx. x− (nnth ls n)) (ndropn n ls)) 0)
using *pfilt-nnth*
by (*metis i0-lb zero-enat-def*)
have *5*: *nnth (ndropn (nnth ls n) nell) (nnth (nmap (λx. x− (nnth ls n)) (ndropn n ls)) 0) =*
nnth (ndropn (nnth ls n) nell) (nnth ls (n+0) − (nnth ls n))
using *zero-enat-def* **by** *force*
have *6*: *nnth (ndropn (nnth ls n) nell) (nnth ls (n+0) − (nnth ls n)) =*
nnth (ndropn (nnth ls n) nell) 0
by *simp*
have *7*: *nnth (ndropn (nnth ls n) nell) 0 = nnth nell (nnth ls n)*
using *assms* **by** *simp*
have *8*: *nlast(pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) =*
nfirst(pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls)))
using *2 4 5 7 3 6* **by** *presburger*
have *10*: *nnth (nfuse (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))*
(pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls)))) i =
(if i ≤ nlength (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))
then nnth (pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) i
else nnth (pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls)))
(i − (the-enat (nlength (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))))))
using *1 8* **by** (*meson nfuse-nnth not-le-imp-less*)
have *11*: *nlength (pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) = n*
using *assms* **by** (*simp add*: *pfilt-nlength*)
have *12*: *i≤n ⟶ nnth (pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) i =*
nnth (ntaken (nnth ls n) nell) (nnth (ntaken n ls) i)

by (*simp add*: *11 pfilt-nnth*)
have *13*: $i \leq n \longrightarrow nnth\ (ntaken\ (nnth\ ls\ n)\ nell)\ (nnth\ (ntaken\ n\ ls\ )\ i) =$
$nnth\ (ntaken\ (nnth\ ls\ n)\ nell)\ (nnth\ ls\ i)$
by (*simp add*: *ntaken-nnth*)
have *15*: $i \leq n \longrightarrow nnth\ (ntaken\ (nnth\ ls\ n)\ nell)\ (nnth\ ls\ i) = nnth\ nell\ (nnth\ ls\ i)$
using *assms* **by** (*simp add*: *nidx-less-eq ntaken-nnth*)
have *16*: $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ ))$
$(i - (the\text{-}enat\ (nlength\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls)))))\ =$
$nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ ))$
$(i - n)$
by (*simp add*: *11*)
have *17*: $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ ))\ (i - n) =$
$nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nnth\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ )\ (i - n))$
using *assms pfilt-nnth*[*of* $i-n$ $(ndropn\ (nnth\ ls\ n)\ nell)$
$(nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ )\ ]$
by (*metis enat-minus-mono1 idiff-enat-enat ndropn-nlength nlength-nmap pfilt-nlength*)
have *18*: $i > n \longrightarrow nnth\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ )\ (i - n) =$
$nnth\ ls\ (n+(i-n)) - (nnth\ ls\ n)$
using *assms*
by (*metis linorder-le-cases ndropn-nnth nfinite-ntaken nlast-nmap nlength-nmap nnth-nmap ntaken-all ntaken-nlast*)
have *19*: $i>n \longrightarrow nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nnth\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ )\ (i - n))$
$= nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ ((nnth\ ls\ i) - (nnth\ ls\ n))$
by (*simp add*: *18*)
have *20* : $i>n \longrightarrow nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ ((nnth\ ls\ i) - (nnth\ ls\ n)) =$
$nnth\ nell\ ((nnth\ ls\ n) + ((nnth\ ls\ i) - (nnth\ ls\ n)))$
using *assms ndropn-nnth* **by** *blast*
have *22*: $i>n \longrightarrow (nnth\ ls\ n) + ((nnth\ ls\ i) - (nnth\ ls\ n)) = (nnth\ ls\ i)$
using *assms* **by** (*simp add*: *nidx-less-eq*)
have *23*: $i>n \longrightarrow nnth\ nell\ ((nnth\ ls\ n) + ((nnth\ ls\ i) - (nnth\ ls\ n))) = nnth\ nell\ (nnth\ ls\ i)$
by (*simp add*: *22*)
from *10* **show** *?thesis*
by (*metis 11 12 13 15 16 17 19 20 23 enat-ord-simps*(*1*) *not-le-imp-less*)
qed

lemma *nfuse-pfilt-nnth-b-alt*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
$\neg nfinite\ nell$
$\neg nfinite\ ls$
$i \leq nlength\ ls$
$n \leq nlength\ ls$
shows $nnth\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
$(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ )))\ i =$
$nnth\ nell\ (nnth\ ls\ i)$
proof −
have *1*: $i \leq nlength(nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
$(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ (ndropn\ n\ ls)\ )))$

using *assms*
by (*metis nfuse-pfilt-nlength pfilt-nlength*)
have *2*: *nlast*(*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) = *nnth nell* (*nnth ls n*)
using *assms pfilt-nnth*[*of* - (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)]
by (*simp add*: *nlength-eq-enat-nfiniteD nnth-nlast ntaken-nnth pfilt-nlength*)
have *3*: *nfirst*(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) =
nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) *0*
by (*metis ndropn-0 ndropn-nfirst*)
have *4*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) *0* =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) *0*)
using *pfilt-nnth*
by (*metis i0-lb zero-enat-def*)
have *5*: *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) *0*) =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth ls* (*n*+*0*) − (*nnth ls n*))
using *zero-enat-def* **by** *force*
have *6*: *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth ls* (*n*+*0*) − (*nnth ls n*)) =
nnth (*ndropn* (*nnth ls n*) *nell*) *0*
by *simp*
have *7*: *nnth* (*ndropn* (*nnth ls n*) *nell*) *0* = *nnth nell* (*nnth ls n*)
using *assms* **by** *simp*
have *8*: *nlast*(*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) =
nfirst(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
using *2 4 5 7 3 6* **by** *presburger*
have *10*: *nnth* (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i* =
(*if i* ≤ *nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
then nnth (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) *i*
else nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − (*the-enat* (*nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))))))
using *1 8* **by** (*meson nfuse-nnth not-le-imp-less*)
have *11*: *nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) = *n*
using *assms* **by** (*simp add*: *pfilt-nlength*)
have *12*: *i*≤*n* ⟶ *nnth* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) *i* =
nnth (*ntaken* (*nnth ls n*) *nell*) (*nnth* (*ntaken n ls*) *i*)
by (*simp add*: *11 pfilt-nnth*)
have *13*: *i*≤*n* ⟶ *nnth* (*ntaken* (*nnth ls n*) *nell*) (*nnth* (*ntaken n ls*) *i*) =
nnth (*ntaken* (*nnth ls n*) *nell*) (*nnth ls i*)
by (*simp add*: *ntaken-nnth*)
have *15*: *i*≤*n* ⟶ *nnth* (*ntaken* (*nnth ls n*) *nell*) (*nnth ls i*) = *nnth nell* (*nnth ls i*)
using *assms* **by** (*simp add*: *nidx-less-eq ntaken-nnth*)
have *16*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − (*the-enat* (*nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))))) =
nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − *n*)
by (*simp add*: *11*)
have *17*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) (*i* − *n*) =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*))
using *assms pfilt-nnth*[*of i*−*n* (*ndropn* (*nnth ls n*) *nell*)
(*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))]
by (*metis enat-minus-mono1 idiff-enat-enat ndropn-nlength nlength-nmap pfilt-nlength*)

have *18*: *i*> *n* ⟶ *nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*) =
nnth ls (*n*+(*i*−*n*)) − (*nnth ls n*)
using *assms*
by (*metis linorder-le-cases ndropn-nnth nfinite-ntaken nlast-nmap nlength-nmap nnth-nmap ntaken-all ntaken-nlast*)
have *19*: *i*>*n* ⟶ *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*))
= *nnth* (*ndropn* (*nnth ls n*) *nell*) ((*nnth ls i*) − (*nnth ls n*))
by (*simp add*: *18*)
have *20* : *i*>*n* ⟶ *nnth* (*ndropn* (*nnth ls n*) *nell*) ((*nnth ls i*) − (*nnth ls n*)) =
nnth nell ((*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*)))
using *assms ndropn-nnth* **by** *blast*
have *22*: *i*>*n* ⟶ (*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*)) = (*nnth ls i*)
using *assms* **by** (*simp add*: *nidx-less-eq*)
have *23*: *i*>*n* ⟶ *nnth nell* ((*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*))) = *nnth nell* (*nnth ls i*)
by (*simp add*: *22*)
from *10* **show** *?thesis*
by (*metis 11 12 13 15 16 17 19 20 23 enat-ord-simps*(*1*) *not-le-imp-less*)
qed

lemma *nfuse-pfilt-nnth*:
assumes *nidx ls*
nnth ls 0 = *0*
(*nlast ls*) = *the-enat*(*nlength nell*)
nfinite nell
nfinite ls
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls*) *i* =
nnth (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*
using *assms nfuse-pfilt-nnth-a*[*of ls nell i n*]
nfuse-pfilt-nnth-b[*of ls nell i n*]
by *simp*

lemma *nfuse-pfilt-nnth-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite nell*
¬*nfinite ls*
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls*) *i* =
nnth (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*
using *assms nfuse-pfilt-nnth-a-alt*[*of ls nell i n*]
nfuse-pfilt-nnth-b-alt[*of ls nell i n*]
by *simp*

lemma *nfuse-pfilt*:
assumes *nidx ls*
nnth ls 0 = 0
(nlast ls) = the-enat(nlength nell)
nfinite nell
nfinite ls
n ≤ nlength ls
shows *pfilt (nfuse (ntaken (nnth ls n) nell) (ndropn (nnth ls n) nell)) ls =*
nfuse (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))
(pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls)))
using *assms nfuse-pfilt-nlength nfuse-pfilt-nnth*
nellist-eq-nnth-eq **by** *(metis pfilt-nlength)*

lemma *nfuse-pfilt-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite nell
¬nfinite ls
n ≤ nlength ls
shows *pfilt (nfuse (ntaken (nnth ls n) nell) (ndropn (nnth ls n) nell)) ls =*
nfuse (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))
(pfilt (ndropn (nnth ls n) nell) (nmap (λx. x− (nnth ls n)) (ndropn n ls)))
using *assms nfuse-pfilt-nlength nfuse-pfilt-nnth-alt*
nellist-eq-nnth-eq **by** *(metis pfilt-nlength)*

lemma *nfuse-pfilt-a*:
assumes *nidx ls1*
nfinite ls1
nnth ls1 0 = 0
nidx ls2
nfinite ls2
nnth ls2 0 = nlast ls1
(nlast ls2) = the-enat(nlength nell)
nfinite nell
shows *pfilt (nfuse (ntaken (nlast ls1) nell) (ndropn (nfirst ls2) nell)) (nfuse ls1 ls2) =*
nfuse (pfilt (ntaken (nlast ls1) nell) ls1)
(pfilt (ndropn (nfirst ls2) nell) (nmap (λx. x− (nfirst ls2)) ls2))
proof −
have *0*: *∃ ll. (enat ll) = nlength ls1*
using *assms(2) nfinite-nlength-enat* **by** *fastforce*
obtain *ll* **where** *01*: *(enat ll) = nlength ls1*
using *0* **by** *auto*
have *1*: *nlast ls1 = nfirst ls2*
using *assms* **by** *(metis nlast-NNil ntaken-0 ntaken-nlast)*
have *2*: *nidx (nfuse ls1 ls2)*
using *assms nidx-nfuse* **by** *blast*
have *20*: *nnth (nfuse ls1 ls2) 0 = 0*
by *(metis 1 assms(3) nfuse-nnth zero-enat-def zero-le)*
have *3*: *nlast(nfuse ls1 ls2) = the-enat(nlength nell)*

```
  using assms
  by (simp add: 1 nlast-nfuse)
 have 4: ll ≤ nlength (nfuse ls1 ls2)
    by (simp add: 01 nfuse-nlength)
 have 5: pfilt (nfuse (ntaken (nnth (nfuse ls1 ls2) ll) nell)
                      (ndropn (nnth (nfuse ls1 ls2) ll) nell))
               (nfuse ls1 ls2)
        =
        nfuse (pfilt (ntaken (nnth (nfuse ls1 ls2) ll) nell) (ntaken ll (nfuse ls1 ls2)))
              (pfilt (ndropn (nnth (nfuse ls1 ls2) ll) nell)
                     (nmap (λ x. x− (nnth (nfuse ls1 ls2) ll)) (ndropn ll (nfuse ls1 ls2))))
  using 2 3 4 nfuse-pfilt[of (nfuse ls1 ls2) nell ll]
  using 20 assms nfuse-nfinite by blast
 have 6: pfilt (nfuse (ntaken (nlast ls1) nell) (ndropn (nfirst ls2) nell)) (nfuse ls1 ls2) =
         pfilt (nfuse (ntaken (nnth (nfuse ls1 ls2) ll) nell)
                      (ndropn (nnth (nfuse ls1 ls2) ll) nell))
               (nfuse ls1 ls2)
   using 1 4
   by (metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps)
 have 7: (pfilt (ntaken (nlast ls1) nell) ls1) =
         (pfilt (ntaken (nnth (nfuse ls1 ls2) ll) nell) (ntaken ll (nfuse ls1 ls2)))
   using 1 4
   by (metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps)
 have 8: (pfilt (ndropn (nfirst ls2) nell) (nmap (λx. x− (nfirst ls2)) ls2)) =
         (pfilt (ndropn (nnth (nfuse ls1 ls2) ll) nell)
                (nmap (λx. x− (nnth (nfuse ls1 ls2) ll)) (ndropn ll (nfuse ls1 ls2)) ))
   using 1 4
   by (metis 01 ndropn-nfirst ndropn-nfuse nfinite-conv-nlength-enat the-enat.simps)
 show ?thesis using 5 6 7 8 by auto
qed


lemma nfuse-pfilt-a-alt:
 assumes nidx  ls1
         nfinite ls1
         nnth ls1 0 = 0
         nidx ls2
         ¬nfinite ls2
         nnth ls2 0 = nlast ls1
         ¬nfinite nell
 shows    pfilt (nfuse (ntaken (nlast ls1) nell) (ndropn (nfirst ls2) nell)) (nfuse ls1 ls2) =
          nfuse (pfilt (ntaken (nlast ls1) nell) ls1)
                (pfilt (ndropn (nfirst ls2) nell) (nmap (λx. x− (nfirst ls2)) ls2))
proof −
 have 0: ∃ ll. (enat ll) = nlength ls1
   using assms(2) nfinite-nlength-enat by fastforce
 obtain ll where 01:  (enat ll) = nlength ls1
     using 0 by auto
 have 1: nlast ls1 = nfirst ls2
  using assms by (metis nlast-NNil ntaken-0 ntaken-nlast)
```

have *2*: *nidx (nfuse ls1 ls2)*
using *assms nidx-nfuse* **by** *blast*
have *20*: *nnth (nfuse ls1 ls2) 0 = 0*
by (*metis 1 assms(3) nfuse-nnth zero-enat-def zero-le*)
have *4*: *ll ≤ nlength (nfuse ls1 ls2)*
by (*simp add*: *01 nfuse-nlength*)
have *5*: *pfilt (nfuse (ntaken (nnth (nfuse ls1 ls2) ll) nell)*
(ndropn (nnth (nfuse ls1 ls2) ll) nell))
(nfuse ls1 ls2)
=
nfuse (pfilt (ntaken (nnth (nfuse ls1 ls2) ll) nell) (ntaken ll (nfuse ls1 ls2)))
(pfilt (ndropn (nnth (nfuse ls1 ls2) ll) nell)
(nmap (λ x. x− (nnth (nfuse ls1 ls2) ll)) (ndropn ll (nfuse ls1 ls2))))
using *2 4 nfuse-pfilt-alt*[*of (nfuse ls1 ls2) nell ll*] *20 assms nfuse-nfinite* **by** *blast*
have *6*: *pfilt (nfuse (ntaken (nlast ls1) nell) (ndropn (nfirst ls2) nell)) (nfuse ls1 ls2) =*
pfilt (nfuse (ntaken (nnth (nfuse ls1 ls2) ll) nell)
(ndropn (nnth (nfuse ls1 ls2) ll) nell))
(nfuse ls1 ls2)
using *1 4*
by (*metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps*)
have *7*: (*pfilt (ntaken (nlast ls1) nell) ls1*) =
(*pfilt (ntaken (nnth (nfuse ls1 ls2) ll) nell) (ntaken ll (nfuse ls1 ls2)*))
using *1 4*
by (*metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps*)
have *8*: (*pfilt (ndropn (nfirst ls2) nell) (nmap (λx. x− (nfirst ls2)) ls2*)) =
(*pfilt (ndropn (nnth (nfuse ls1 ls2) ll) nell)*
(nmap (λx. x− (nnth (nfuse ls1 ls2) ll)) (ndropn ll (nfuse ls1 ls2))))
using *1 4*
by (*metis 01 ndropn-nfirst ndropn-nfuse nfinite-conv-nlength-enat the-enat.simps*)
show *?thesis* **using** *5 6 7 8* **by** *auto*
qed

lemma *pfilt-ntaken*:
assumes *n ≤ nlength ls*
shows *ntaken n (pfilt σ ls) = pfilt σ (ntaken n ls)*
proof −
have *1*: *nlength (ntaken n (pfilt σ ls)) = nlength (pfilt σ (ntaken n ls))*
by (*simp add*: *assms pfilt-nlength*)
have *2*: ∀ (*i*::*nat*). *i*≤*nlength*(*ntaken n (pfilt σ ls*))⟶
(*nnth (ntaken n (pfilt σ ls)) i*) = (*nnth (pfilt σ (ntaken n ls)) i*)
by (*simp add*: *pfilt-nmap*)
show *?thesis* **using** *1 2 nellist-eq-nnth-eq* **by** *blast*
qed

lemma *pfilt-ntaken-nidx*:
assumes *n ≤ nlength nell*
nidx ls
nnth ls 0 = 0
nfinite ls

nlast ls = n
shows *pfilt nell ls = pfilt (ntaken n nell) ls*
proof −
have *1*: *nlength(pfilt nell ls) = nlength (pfilt (ntaken n nell) ls)*
by (*simp add*: *pfilt-nlength*)
have *2*: ∀(*i*::*nat*). *i*≤*nlength*(*pfilt nell ls*) ⟶
(*nnth* (*pfilt nell ls*) *i*) = (*nnth* (*pfilt* (*ntaken n nell*) *ls*) *i*)
using *assms*
by (*auto simp add*: *nidx-all-le-nlast ntaken-nnth pfilt-nlength pfilt-nnth*)
show *?thesis*
by (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

lemma *pfilt-ndropn*:
assumes *n* ≤ *nlength ls*
shows *ndropn n* (*pfilt nell ls*) = *pfilt nell* (*ndropn n ls*)
proof −
have *1*: *nlength* (*ndropn n* (*pfilt nell ls*)) = *nlength* (*pfilt nell* (*ndropn n ls*))
by (*simp add*: *assms pfilt-nlength*)
have *2*: ∀ (*i*::*nat*). *i*≤*nlength* (*ndropn n* (*pfilt nell ls*)) ⟶
(*nnth* (*ndropn n* (*pfilt nell ls*)) *i*) = (*nnth* (*pfilt nell* (*ndropn n ls*)) *i*)
by (*simp add*: *ndropn-nmap pfilt-nmap*)
show *?thesis* **using** *1 2 nellist-eq-nnth-eq* **by** *blast*
qed

lemma *pfilt-ndropn-nidx-nlength*:
assumes (*enat n*) ≤ *nlength nell*
nidx ls
nnth ls 0 = 0
shows *nlength* (*pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*)) = *nlength* (*pfilt* (*ndropn n nell*) *ls*)
using *assms* **by** (*simp add*: *pfilt-nlength*)

lemma *pfilt-ndropn-nidx-nnth*:
assumes (*enat n*) ≤ *nlength nell*
nidx ls
nnth ls 0 = 0
i≤*nlength ls*
shows *nnth* (*pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*)) *i* = *nnth* (*pfilt* (*ndropn n nell*) *ls*) *i*
proof −
have *1*: *nnth* (*pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*)) *i* = *nnth nell* (*nnth* (*nmap* (λ*x*. *x*+ *n*) *ls*) *i*)
by (*simp add*: *assms pfilt-nnth pfilt-nlength*)
have *2*: *nnth nell* (*nnth* (*nmap* (λ*x*. *x*+ *n*) *ls*) *i*) = (*nnth nell* ((*nnth ls i*) + *n*))
by (*simp add*: *assms*)
have *3*: *nnth* (*pfilt* (*ndropn n nell*) *ls*) *i* = *nnth* (*ndropn n nell*) (*nnth ls i*)
by (*simp add*: *assms pfilt-nnth pfilt-nlength*)
have *4*: *nnth* (*ndropn n nell*) (*nnth ls i*) = (*nnth nell* ((*nnth ls i*) + *n*))
by (*metis ntaken-ndropn-nlast ntaken-nlast*)
show *?thesis*

using *1 2 3 4* **by** *presburger*
qed

lemma *pfilt-ndropn-nidx*:
assumes (*enat n*) ≤ *nlength nell*
nidx ls
nnth ls 0 = 0
shows *pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*) = *pfilt* (*ndropn n nell*) *ls*
using *assms pfilt-ndropn-nidx-nlength pfilt-ndropn-nidx-nnth nellist-eq-nnth-eq*
by (*simp add*: *pfilt-ndropn-nidx-nnth nellist-eq-nnth-eq pfilt-nlength*)

lemma *pfilt-pfilt*:
(*nnth* (*pfilt* (*pfilt nell ls1*) *ls2*) *k*) = (*nnth nell* (*nnth ls1* (*nnth ls2 k*)))
by (*metis enat-le-plus-same*(*1*) *gen-nlength-def ndropn-nmap nlength-code nnth-nmap nnth-zero-ndropn pfilt-nmap*)

lemma *pfilt-pfilt-nmap*:
(*pfilt* (*pfilt nell ls1*) *ls2*) = (*pfilt nell* (*nmap* (λ *x*. (*nnth ls1 x*)) *ls2*))
by (*simp add*: *pfilt-expand pfilt-nlength pfilt-pfilt*)

lemma *pfilt-nmap-pfilt*:
(*pfilt* (*pfilt nell ls1*) *ls2*) = *pfilt nell* (*pfilt ls1 ls2*)
by (*metis pfilt-nmap pfilt-pfilt-nmap*)

lemma *pfilt-nsubn*:
assumes *k* ≤ *n*
n ≤ *nlength ls*
shows (*nsubn* (*pfilt nell ls*)) *k n* = (*pfilt nell* (*nsubn ls k n*))
using *assms*
by (*simp add*: *ndropn-nmap nsubn-def1 pfilt-nmap*)

lemma *pfilt-nfusecat-nmap*:
(*pfilt nell* (*nfusecat lls*)) = (*nfusecat* (*nmap* (λ *ls* . (*pfilt nell ls*)) *lls*))
proof −
have *1*: *pfilt nell* (*nfusecat lls*) = *nmap* (*nnth nell*) (*nfusecat lls*)
using *pfilt-nmap*[*of nell* (*nfusecat lls*)] **by** *blast*
have *2*: (*nfusecat* (*nmap* (λ *ls* . (*pfilt nell ls*)) *lls*)) =
(*nfusecat* (*nmap* (*nmap* (*nnth nell*)) *lls*))
using *pfilt-nmap* **by** *metis*
have *3*: *nmap* (*nnth nell*) (*nfusecat lls*) = (*nfusecat* (*nmap* (*nmap* (*nnth nell*)) *lls*))
by (*simp add*: *nmap-nfusecat*)
show *?thesis*
by (*simp add*: *1 2 3*)
qed

powerinterval lemmas

lemma *powerinterval-splita0*:

assumes *nidx ls*
nnth ls 0 = 0
n ≤ nlength ls
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
i < nlength(ntaken n ls)
shows *(nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) = (nsubn σ (nnth ls i) (nnth ls (Suc i)))*
proof −
have *01*: *(nnth ls n) ≤ nlength σ*
using *assms*
by (*metis enat-ord-code(4) enat-the-enat finite-nidx-bounded-nlast order-less-imp-le*)
have *02*: *(nnth (ntaken n ls) i) ≤ (nnth (ntaken n ls) (Suc i))*
using *assms*
by (*metis dual-order.trans eSuc-enat enat-ord-simps(1) ileI1 le-add2 min-def nidx-less-eq ntaken-nlength ntaken-nnth plus-1-eq-Suc*)
have *03*: *(nnth (ntaken n ls) (Suc i)) ≤ (nnth ls n)*
using *assms*
by (*metis eSuc-enat enat-ord-simps(1) ileI1 min-def nidx-less-eq ntaken-nlength ntaken-nnth*)
show *?thesis* **unfolding** *nsubn-def1* **using** *01 02 03 assms*
by (*simp add*: *ntaken-nnth ntaken-ndropn-swap order-subst2*)
qed

lemma *powerinterval-splita0-alt*:
assumes *nidx ls*
nnth ls 0 = 0
n ≤ nlength ls
¬nfinite ls
¬nfinite σ
i < nlength(ntaken n ls)
shows *(nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) = (nsubn σ (nnth ls i) (nnth ls (Suc i)))*
proof −
have *01*: *(nnth ls n) ≤ nlength σ*
by (*meson assms(5) enat-less-imp-le less-enatE nfinite-conv-nlength-enat*)
have *02*: *(nnth (ntaken n ls) i) ≤ (nnth (ntaken n ls) (Suc i))*
using *assms*
by (*metis dual-order.trans eSuc-enat enat-ord-simps(1) ileI1 le-add2 min-def nidx-less-eq ntaken-nlength ntaken-nnth plus-1-eq-Suc*)
have *03*: *(nnth (ntaken n ls) (Suc i)) ≤ (nnth ls n)*
using *assms*
by (*metis eSuc-enat enat-ord-simps(1) ileI1 min-def nidx-less-eq ntaken-nlength ntaken-nnth*)
show *?thesis* **unfolding** *nsubn-def1* **using** *01 02 03 assms*
by (*simp add*: *ntaken-nnth ntaken-ndropn-swap order-subst2*)
qed

lemma *powerinterval-splita*:
assumes *nidx ls*
nnth ls 0 = 0

```
        nfinite ls
         n ≤ nlength ls
        nfinite σ
         nlast ls  = the-enat(nlength σ)
         powerinterval f σ ls
 shows    powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls)
proof −
 have 0: (∀ i.  i<nlength ls ⟶ ( (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f))
   using assms by (simp add: powerinterval-def)
 have 1:  powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) =
          (∀ i. i < nlength(ntaken n ls) ⟶
               ((nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f))
       using powerinterval-def by blast
 have 2: (∀ i. i < nlength(ntaken n ls) ⟶
               ((nsubn (ntaken (nnth ls n) σ)  (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f))
  proof
    fix i
    show i < nlength(ntaken n ls) ⟶
               ((nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
    proof −
     have 21:  i<n ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
       using 0 assms less-le-trans by (metis enat-ord-simps(2))
     have 22: i<nlength(ntaken n ls) ⟶
              ((nsubn σ (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
       by (simp add: 21 assms(4) ntaken-nnth)
     have 23: i<nlength(ntaken n ls) ⟶
              ((nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
      using 22 assms powerinterval-splita0
      using 21 by fastforce
     show ?thesis using 23 by blast
    qed
   qed
  show ?thesis using 1 2 by blast
qed

lemma powerinterval-splita-alt:
 assumes  nidx ls
         nnth ls 0 = 0
         ¬nfinite ls
          n ≤ nlength ls
         ¬nfinite σ
          powerinterval f σ ls
 shows    powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls)
proof −
 have 0: (∀ i.  i<nlength ls ⟶ ( (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f))
   using assms by (simp add: powerinterval-def)
 have 1:  powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) =
          (∀ i. i < nlength(ntaken n ls) ⟶
               ((nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f))
       using powerinterval-def by blast
```

```
 have 2: (∀ i. i < nlength(ntaken n ls) ⟶
             ((nsubn (ntaken (nnth ls n) σ)  (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f))
  proof
    fix i
    show i < nlength(ntaken n ls) ⟶
             ((nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
    proof −
     have 21:  i<n ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
       using 0 assms less-le-trans by (metis enat-ord-simps(2))
     have 22: i<nlength(ntaken n ls) ⟶
             ((nsubn σ (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
       by (simp add: 21 assms(4) ntaken-nnth)
     have 23: i<nlength(ntaken n ls) ⟶
             ((nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
      using 22 assms powerinterval-splita0-alt
      using 21 by fastforce
     show ?thesis using 23 by blast
    qed
   qed
  show ?thesis using 1 2 by blast
qed

lemma powerinterval-splitb0:
 assumes  nidx  ls
         nnth ls 0 = 0
         nfinite ls
          n ≤ nlength ls
         nfinite σ
          nlast ls =  the-enat(nlength σ)
           i < (nlength ls −n)
 shows   (nsubn (ndropn (nnth ls n) σ)
              (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) i)
              (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) (Suc i))
                ) =
          (nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) )
proof −
 have 1: nlength (((nmap (λx. x−  (nnth ls n))  (ndropn n ls)))) = (nlength ls) − n
     by (simp add: assms)
 have 2:  i<(nlength ls −n) ⟶
            (nnth (((nmap (λx .x− (nnth ls n))  (ndropn n ls)))) i) =
             (nnth ls (n + i)) − (nnth ls n)
          by simp
 have 3: i<(nlength ls −n) ⟶
            (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) (Suc i)) =
            (nnth ls (n + (Suc i))) − (nnth ls n)
      using assms by (metis eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap)
 have 4: i<(nlength ls −n) ⟶
            (nsubn (ndropn (nnth ls n) σ)
                 (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) i)
                 (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) (Suc i))
```

```
                ) =
              (nsubn (ndropn (nnth ls n) σ)
                  ( (nnth ls (n + i)) − (nnth ls n) )
                  ( (nnth ls (n + (Suc i))) − (nnth ls n) )
                )
         by (simp add: 2 3)
 have 5:  i<(nlength ls −n) ⟶ (nnth ls (n + i))  < (nnth ls (n + (Suc i)))
  using assms
  by (metis Suc-ile-eq add.commute add-Suc-right enat-min nidx-expand plus-enat-simps(1))
 have 6: i<(nlength ls −n) ⟶ (nnth ls n) ≤ (nnth ls (n + i))
    using assms
    by (metis add.commute enat-min le-add1 nidx-less-eq order-less-imp-le plus-enat-simps(1))
 have 7: i<(nlength ls −n) ⟶ (nnth ls n) ≤ (nnth ls (n + (Suc i)))
    using 5 6 by linarith
 have 8:  i<(nlength ls −n) ⟶ (nnth ls (n + i)) − (nnth ls n) < (nnth ls (n + (Suc i))) − (nnth ls n)
   using 5 6 diff-less-mono by blast
 have 10: i<(nlength ls −n) ⟶
             (nsubn (ndropn (nnth ls n) σ)
                  ( (nnth ls (n + i)) − (nnth ls n) )
                  ( (nnth ls (n + (Suc i))) − (nnth ls n) )
                ) =
              (nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) )
   by (metis 6 7 8 add.commute diff-add nsubn-ndropn)
 show ?thesis
 using 2 3 10 assms by auto
qed

lemma powerinterval-splitb0-alt:
 assumes  nidx  ls
         nnth ls 0 = 0
         ¬nfinite ls
          n ≤ nlength ls
         ¬nfinite σ
           i < (nlength ls −n)
 shows   (nsubn (ndropn (nnth ls n) σ)
                (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) i)
                (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) (Suc i))
                  ) =
          (nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) )
proof −
 have 1: nlength (((nmap (λx. x−  (nnth ls n))  (ndropn n ls)))) = (nlength ls) − n
     by (simp add: assms)
 have 2:  i<(nlength ls −n) ⟶
              (nnth (((nmap (λx .x− (nnth ls n))  (ndropn n ls)))) i) =
               (nnth ls (n + i)) − (nnth ls n)
          by simp
 have 3: i<(nlength ls −n) ⟶
              (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) (Suc i)) =
              (nnth ls (n + (Suc i))) − (nnth ls n)
       using assms by (metis eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap)
```

have *4*: *i*<(*nlength ls* −*n*) ⟶
(*nsubn* (*ndropn* (*nnth ls n*) *σ*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) =
(*nsubn* (*ndropn* (*nnth ls n*) *σ*)
((*nnth ls* (*n* + *i*)) − (*nnth ls n*))
((*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*))
)
by (*simp add*: *2 3*)
have *5*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls* (*n* + *i*)) < (*nnth ls* (*n* + (*Suc i*)))
using *assms*
by (*metis Suc-ile-eq add.commute add-Suc-right enat-min nidx-expand plus-enat-simps*(*1*))
have *6*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls n*) ≤ (*nnth ls* (*n* + *i*))
using *assms*
by (*metis add.commute enat-min le-add1 nidx-less-eq order-less-imp-le plus-enat-simps*(*1*))
have *7*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls n*) ≤ (*nnth ls* (*n* + (*Suc i*)))
using *5 6* **by** *linarith*
have *8*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls* (*n* + *i*)) − (*nnth ls n*) < (*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*)
using *5 6 diff-less-mono* **by** *blast*
have *9*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*) ≤ *nlength σ* − (*nnth ls n*)
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *10*: *i*<(*nlength ls* −*n*) ⟶
(*nsubn* (*ndropn* (*nnth ls n*) *σ*)
((*nnth ls* (*n* + *i*)) − (*nnth ls n*))
((*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*))
) =
(*nsubn σ* (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*)))
by (*metis 6 7 8 add.commute diff-add nsubn-ndropn*)
show *?thesis*
using *2 3 10 assms* **by** *auto*
qed

lemma *powerinterval-splitb*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
n ≤ *nlength ls*
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
powerinterval f σ ls
shows *powerinterval f* (*ndropn* (*nnth ls n*) *σ*) ((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
proof −
have *0*: (∀ *i*. *i*<*nlength ls* ⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f*))
using *assms* **by** (*simp add*: *powerinterval-def*)
have *1*: *powerinterval f* (*ndropn* (*nnth ls n*) *σ*) ((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) =
(∀ *i*. *i*<*nlength* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟶
((*nsubn* (*ndropn* (*nnth ls n*) *σ*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)

```
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) (Suc i))
                ) ⊨ f))
    by (simp add: powerinterval-def)
 have 2: (∀ i. i<nlength(((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) ⟶
          ( (nsubn (ndropn (nnth ls n) σ)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) i)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) (Suc i))
                ) ⊨ f))
  proof
   fix i
   show i<nlength(((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) ⟶
          ( (nsubn (ndropn (nnth ls n) σ)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) i)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) (Suc i))
                ) ⊨ f)
   proof −
    have 21: i< (nlength ls) −n ⟶
             ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) ) ⊨ f)
       using 0 assms(4) enat-min by auto
    show ?thesis
    using 21 assms powerinterval-splitb0 by fastforce
   qed
  qed
 show ?thesis using 1 2 by blast
qed

lemma powerinterval-splitb-alt:
 assumes nidx ls
         nnth ls 0 = 0
         ¬nfinite ls
           n ≤ nlength ls
         ¬nfinite σ
          powerinterval f σ ls
 shows    powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n)) (ndropn n ls)))
proof −
 have 0: (∀ i. i<nlength ls ⟶ ( (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f))
   using assms by (simp add: powerinterval-def)
 have 1: powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n)) (ndropn n ls))) =
         (∀ i. i<nlength (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) ⟶
           ( (nsubn (ndropn (nnth ls n) σ)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) i)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) (Suc i))
                ) ⊨ f))
    by (simp add: powerinterval-def)
 have 2: (∀ i. i<nlength(((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) ⟶
          ( (nsubn (ndropn (nnth ls n) σ)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) i)
                (nnth (((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) (Suc i))
                ) ⊨ f))
  proof
```

```
  fix i
  show i<nlength(((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) ⟶
         ( (nsubn (ndropn (nnth ls n) σ)
                (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) i)
                (nnth (((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) (Suc i))
                ) ⊨ f)
  proof −
   have 21:  i< (nlength ls) −n ⟶
           ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) ) ⊨ f)
      using 0 assms(4) enat-min by auto
   show ?thesis
   using 21 assms powerinterval-splitb0-alt by fastforce
  qed
 qed
 show ?thesis using 1 2 by blast
qed

lemma powerinterval-split:
 assumes  nidx  ls
        nnth ls 0 = 0
        nfinite ls
          n ≤ nlength ls
        nfinite σ
         nlast ls =  the-enat(nlength σ)
 shows    powerinterval f σ ls =
          (powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) ∧
           powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n))  (ndropn n ls))))
proof −
 have 1: powerinterval f σ ls ⟹
         powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls)
   by (simp add: assms powerinterval-splita)
 have 2: powerinterval f σ ls ⟹
         powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n))  (ndropn n ls)))
   by (simp add: assms powerinterval-splitb)
 have 3: powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) =
         (∀ i. i<n ⟶ ( (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f))
    using assms by (simp add:  powerinterval-def powerinterval-splita0)
 have 4: powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n))  (ndropn n ls))) =
        (∀  i. i<(nlength ls) −n ⟶ ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) ) ⊨ f))
    unfolding powerinterval-def using assms powerinterval-splitb0[of ls n σ] by (simp)
 have 5: (∀  i. i<(nlength ls) −n ⟶ ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) ) ⊨ f)) =
         (∀  i. n ≤ i ∧ i<(nlength ls) ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f))
     using assms  enat-min by auto
        (metis diff-add diff-less-mono enat-ord-simps(2) idiff-enat-enat nfinite-nlength-enat)
 have 6: (powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) ∧
           powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n))  (ndropn n ls)))) ⟹
          powerinterval f σ ls
  using 3 4 5 assms powerinterval-def
  by (metis not-less-eq not-less-less-Suc-eq order.order-iff-strict)
 show ?thesis
```

using *1 2 6* **by** *blast*
qed

lemma *powerinterval-split-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
n ≤ nlength ls
¬nfinite σ
shows *powerinterval f σ ls =*
(powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) ∧
powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n)) (ndropn n ls))))
proof −
have *1*: *powerinterval f σ ls ⟹*
powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls)
by (*simp add*: *assms powerinterval-splita-alt*)
have *2*: *powerinterval f σ ls ⟹*
powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n)) (ndropn n ls)))
by (*simp add*: *assms powerinterval-splitb-alt*)
have *3*: *powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) =*
(∀ i. i<n ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f))
using *assms* **by** (*simp add*: *powerinterval-def powerinterval-splita0-alt*)
have *4*: *powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n)) (ndropn n ls))) =*
(∀ i. i<(nlength ls) −n ⟶ ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n))) ⊨ f))
unfolding *powerinterval-def* **using** *powerinterval-splitb0-alt*[*of ls n σ*] *assms*
by *simp*
have *5*: *(∀ i. i<(nlength ls) −n ⟶ ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n))) ⊨ f)) =*
(∀ i. n ≤ i ∧ i<(nlength ls) ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f))
using *assms enat-min*
by *auto*
(*metis diff-add ndropn-nlength nfinite-ndropn-b nfinite-ntaken not-le-imp-less ntaken-all*)
have *6*: (*powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) ∧*
powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n)) (ndropn n ls)))) ⟹
powerinterval f σ ls
using *3 4 5 assms powerinterval-def*
by (*metis not-less-eq not-less-less-Suc-eq order.order-iff-strict*)
show *?thesis*
using *1 2 6* **by** *blast*
qed

lemma *powerinterval-nfuse*:
assumes *nidx ls1*
nnth ls1 0 = 0
nidx ls2
nnth ls2 0 = 0
nfinite ls1
nlast ls1 = cp
cp ≤ nlength σ
nfinite ls2
nfinite σ

```
        nlast ls2 = the-enat(nlength σ) − cp
 shows   powerinterval f σ (nfuse ls1 (nmap (λx. x+ cp) ls2)) =
         (powerinterval f (ntaken cp σ) ls1 ∧
          powerinterval f (ndropn cp σ) ls2 )
proof −
 have 1: nidx (nfuse ls1 (nmap (λx. x+ cp) ls2))
  using assms nidx-nfuse[of ls1 (nmap (λx. x+ cp) ls2)]
   using Suc-ile-eq nidx-expand zero-enat-def by force
 have 2: cp = (nnth ls1 (the-enat (nlength ls1)))
  using assms using nnth-nlast by blast
 have 3: nlength ls1 ≤ nlength (nfuse ls1 (nmap (λx. x+ cp) ls2))
  by (simp add: nfuse-nlength)
 have 30: nfinite (nfuse ls1 (nmap (λx. x+ cp) ls2))
  using assms
 by (metis nfinite-conv-nlength-enat nfuse-nlength nlength-nmap plus-enat-simps(1))
 have 31: nfirst (nmap (λx. x+ cp) ls2) = nnth (nmap (λx. x+ cp) ls2) 0
    by (metis ndropn-0 ndropn-nfirst)
 have 32: nnth (nmap (λx. x+ cp) ls2) 0 = cp
  using assms nnth-nmap[of 0 ls2 (λx. x+ cp)]
  using zero-enat-def by auto
 have 33: nlast ls1 = nfirst (nmap (λx. x+ cp) ls2)
    by (simp add: 31 32 assms(6))
 have 34: nlast (nmap (λx. x + cp) ls2) = the-enat(nlength σ)
  using assms
  by (metis diff-add enat.simps(3) enat-ord-simps(1) enat-the-enat nfinite-conv-nlength-enat nlast-nmap)
 have 4: nlast (nfuse ls1 (nmap (λx. x+ cp) ls2)) = the-enat(nlength σ)
    using assms using 30 assms nlast-nfuse[of ls1 (nmap (λx. x+ cp) ls2)] 33 34 by presburger
 have 5: cp = nnth (nfuse ls1 (nmap (λx. x+ cp) ls2)) (the-enat (nlength ls1))
  using assms 2
 by (metis enat.simps(3) enat-the-enat nfinite-conv-nlength-enat nfuse-def1 nnth-nappend
     not-le-imp-less order-less-imp-le)
 have 50: nlength (nmap (λx. x− cp) (nmap (λx. x+ cp) ls2)) = nlength ls2
   by simp
 have 51: ⋀j. j ≤ nlength ls2 ⟶
              nnth (nmap (λx. x− cp) (nmap (λx. x+ cp) ls2)) j = nnth ls2 j
    by simp
 have 6: (nmap (λx. x− cp) (nmap (λx. x+ cp) ls2)) = ls2
  using 50 51 by (simp add: nellist-eq-nnth-eq)
 have 70: ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap (λx::nat. x + cp) ls2)) = ls1
  by (simp add: 33 assms(5) ntaken-nfuse)
 have 71: ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap (λx::nat. x + cp) ls2)) =
          nmap (λx::nat. x + cp) ls2
  by (simp add: 33 assms(5) ndropn-nfuse)
 have 72: nidx (nfuse (ls1::nat nellist) (nmap (λx::nat. x + (cp::nat)) (ls2::nat nellist)))
  by (simp add: 1)
 have 73: nnth (nfuse ls1 (nmap (λx::nat. x + cp) ls2)) (0::nat) = (0::nat)
      by (metis 33 assms(2) nfuse-nnth zero-enat-def zero-le)
 have 74: enat (the-enat (nlength ls1)) ≤ nlength (nfuse ls1 (nmap (λx::nat. x + cp) ls2))
  by (metis 70 assms(5) enat.simps(3) enat-the-enat min-def nfinite-nlength-enat ntaken-nlength
      order-eq-refl)
```

have *75*: *enat* (*nlast* (*nfuse ls1* (*nmap* (λ*x*::*nat*. *x* + *cp*) *ls2*))) = *nlength* σ
using *4 assms enat-the-enat nfinite-conv-nlength-enat* **by** *auto*
have *76*: *nfinite* σ
by (*simp add*: *assms*(*9*))
have *80*: *powerinterval f* σ (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) =
(*powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*))) ∧
powerinterval f (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λ*x*. *x* − *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)))))
using *72 73 30 74 76 75 4* **using** *powerinterval-split*[*of* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
(*the-enat* (*nlength ls1*)) σ *f*]
by *fastforce*
have *81*: *powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*))) =
powerinterval f (*ntaken cp* σ) *ls1*
using *5 70* **by** *auto*
have *82*: *powerinterval f* (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λ*x*. *x* − *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)))) =
powerinterval f (*ndropn cp* σ) *ls2*
using *5 6 71* **by** *presburger*
show *?thesis*
using *80 81 82* **by** *blast*
qed

lemma *powerinterval-nfuse-alt*:
assumes *nidx ls1*
nnth ls1 0 = *0*
nidx ls2
nnth ls2 0 = *0*
nfinite ls1
nlast ls1 = *cp*
cp ≤ *nlength* σ
¬*nfinite ls2*
¬*nfinite* σ
shows *powerinterval f* σ (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) =
(*powerinterval f* (*ntaken cp* σ) *ls1* ∧
powerinterval f (*ndropn cp* σ) *ls2*)
proof −
have *1*: *nidx* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
using *assms nidx-nfuse*[*of ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)]
using *Suc-ile-eq nidx-expand zero-enat-def* **by** *force*
have *2*: *cp* = (*nnth ls1* (*the-enat* (*nlength ls1*)))
using *assms* **using** *nnth-nlast* **by** *blast*
have *3*: *nlength ls1* ≤ *nlength* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
by (*simp add*: *nfuse-nlength*)
have *30*: ¬*nfinite* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
by (*metis assms*(*8*) *enat-ile enat-le-plus-same*(*2*) *nfinite-conv-nlength-enat nfuse-nlength nlength-nmap*)

have *5*: *cp* = *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) (*the-enat* (*nlength ls1*))
by (*metis 2 assms*(*5*) *enat.simps*(*3*) *enat-the-enat nfinite-conv-nlength-enat nfuse-def1 nnth-nappend not-le-imp-less order-less-imp-le*)
have *50*: *nlength* (*nmap* (λ*x*. *x*− *cp*) (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) = *nlength ls2*
by *simp*
have *51*: ⋀*j*. *j* ≤ *nlength ls2* ⟶
nnth (*nmap* (λ*x*. *x*− *cp*) (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) *j* = *nnth ls2 j*
by *simp*
have *6*: (*nmap* (λ*x*. *x*− *cp*) (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) = *ls2*
using *50 51* **by** (*simp add*: *nellist-eq-nnth-eq*)
have *7*: *nlast ls1* = *nfirst* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)
by (*metis add-0 assms*(*4*) *assms*(*6*) *nfinite-conv-nlength-enat nfinite-ntaken nlast-NNil nlength-NNil nnth-nmap ntaken-0 ntaken-nlast the-enat.simps the-enat-0 zero-le*)
have *70*: *ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*::*nat*. *x* + *cp*) *ls2*)) = *ls1*
by (*simp add*: *7 assms*(*5*) *ntaken-nfuse*)
have *71*: *ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*::*nat*. *x* + *cp*) *ls2*)) =
nmap (λ*x*::*nat*. *x* + *cp*) *ls2*
by (*simp add*: *7 assms*(*5*) *ndropn-nfuse*)
have *72*: *nidx* (*nfuse* (*ls1*::*nat nellist*) (*nmap* (λ*x*::*nat*. *x* + (*cp*::*nat*)) (*ls2*::*nat nellist*)))
by (*simp add*: *1*)
have *73*: *nnth* (*nfuse ls1* (*nmap* (λ*x*::*nat*. *x* + *cp*) *ls2*)) (*0*::*nat*) = (*0*::*nat*)
by (*metis 7 assms*(*2*) *nfuse-nnth zero-enat-def zero-le*)
have *74*: *enat* (*the-enat* (*nlength ls1*)) ≤ *nlength* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*))
by (*metis 70 assms*(*5*) *enat.simps*(*3*) *enat-the-enat min-def nfinite-nlength-enat ntaken-nlength order-eq-refl*)
have *76*: ¬*nfinite* σ
by (*simp add*: *assms*(*9*))
have *80*: *powerinterval f* σ (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) =
(*powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*))) ∧
powerinterval f (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λ*x*::*nat*. *x* − *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)))))
using *72 73 30 74 76*
using *powerinterval-split-alt*[*of* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
(*the-enat* (*nlength ls1*)) σ *f*]
by *blast*
have *81*: *powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*))) =
powerinterval f (*ntaken cp* σ) *ls1*
using *5 70* **by** *auto*
have *82*: *powerinterval f* (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λ*x*. *x* − *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)))) =
powerinterval f (*ndropn cp* σ) *ls2*
using *5 6 71* **by** *presburger*
show *?thesis*
using *80 81 82* **by** *blast*
qed

cppl lemmas

lemma *cppl-expand*:
assumes ($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0 = 0* $\wedge$ *powerinterval f* σ *ls* $\wedge$
((*nfinite ls* $\wedge$ *nfinite* σ $\wedge$ *nlast ls = the-enat*(*nlength* σ)) $\vee$
($\neg$*nfinite ls* $\wedge$ $\neg$ *nfinite* σ)) $\wedge$
((*pfilt* σ *ls*) $\models$ *g*))
shows *nidx* (*cppl f g* σ) $\wedge$ *nnth* (*cppl f g* σ) *0 = 0* $\wedge$ *powerinterval f* σ (*cppl f g* σ) $\wedge$
((*nfinite* (*cppl f g* σ) $\wedge$ *nfinite* σ $\wedge$ *nlast* (*cppl f g* σ) = *the-enat*(*nlength* σ)) $\vee$
($\neg$ *nfinite* (*cppl f g* σ) $\wedge$ $\neg$*nfinite* σ)) $\wedge$ ((*pfilt* σ (*cppl f g* σ)) $\models$ *g*)
proof −
have *0*: *cppl f g* σ = (ϵ *ls. nidx ls* $\wedge$ *nnth ls 0 = 0* $\wedge$ *powerinterval f* σ *ls* $\wedge$
((*nfinite ls* $\wedge$ *nfinite* σ $\wedge$ *nlast ls* = *the-enat*(*nlength* σ)) $\vee$
($\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ)) $\wedge$ ((*pfilt* σ *ls*) $\models$ *g*))
using *cppl-def* **by** *blast*
have *1*: ($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0 = 0* $\wedge$ *powerinterval f* σ *ls* $\wedge$
((*nfinite ls* $\wedge$ *nfinite* σ $\wedge$ *nlast ls* = *the-enat*(*nlength* σ)) $\vee$
($\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ)) $\wedge$ ((*pfilt* σ *ls*) $\models$ *g*))
using *assms* **by** *auto*
have *2*: *nidx* (*cppl f g* σ) $\wedge$ *nnth* (*cppl f g* σ) *0 = 0* $\wedge$ *powerinterval f* σ (*cppl f g* σ) $\wedge$
((*nfinite* (*cppl f g* σ) $\wedge$ *nfinite* σ $\wedge$ *nlast* (*cppl f g* σ) = *the-enat*(*nlength* σ)) $\vee$
($\neg$*nfinite* (*cppl f g* σ) $\wedge$ $\neg$*nfinite* σ)) $\wedge$ ((*pfilt* σ (*cppl f g* σ)) $\models$ *g*)
using *0 1*
someI-ex[*of* λ*ls. nidx ls* $\wedge$ *nnth ls 0 = 0* $\wedge$ *powerinterval f* σ *ls* $\wedge$
((*nfinite ls* $\wedge$ *nfinite* σ $\wedge$ *nlast ls = the-enat*(*nlength* σ)) $\vee$
($\neg$*nfinite ls* $\wedge$ $\neg$ *nfinite* σ)) $\wedge$
((*pfilt* σ *ls*) $\models$ *g*)] **by** *simp*
show *?thesis*
using *2* **by** *blast*
qed

lemma *cppl-fprojection*:
(*σ* $\models$ *f fproj g*) =
(*nidx* (*cppl f g* σ) $\wedge$ *nnth* (*cppl f g* σ) *0 = 0* $\wedge$ *powerinterval f* σ (*cppl f g* σ) $\wedge$
nfinite (*cppl f g* σ) $\wedge$ *nfinite* σ $\wedge$
nlast (*cppl f g* σ) = *the-enat*(*nlength* σ) $\wedge$ *g* (*pfilt* σ (*cppl f g* σ)))
using *cppl-expand*[*of f* σ *g*]
by (*simp add*: *fprojection-d-def*)
(*metis nfinite-conv-nlength-enat the-enat.simps*)

lemma *cppl-oprojection*:
(*σ* $\models$ *f oproj g*) =
(*nidx* (*cppl f g* σ) $\wedge$ *nnth* (*cppl f g* σ) *0 = 0* $\wedge$ *powerinterval f* σ (*cppl f g* σ) $\wedge$
$\neg$*nfinite* (*cppl f g* σ) $\wedge$ $\neg$*nfinite* σ $\wedge$ *g* (*pfilt* σ (*cppl f g* σ)))
using *cppl-expand* **by** (*simp add*: *oprojection-d-def*, *blast*)

lemma *cppl-empty*:
assumes *nlength* σ = *0*
(*σ* $\models$ *f fproj g*)

shows $(cppl\ f\ g\ \sigma) = (NNil\ 0)$
using *assms cppl-fprojection*[*of f g* σ]
by (*metis gr-zeroI less-numeral-extra*(*3*) *ndropn-0 ndropn-nlast nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps the-enat-0*)

lemma *cppl-empty-a*:
assumes *nlength* $\sigma = 0$
$(cppl\ f\ g\ \sigma) = NNil\ 0$
$g(pfilt\ \sigma\ (NNil\ 0))$
shows $(\sigma \models f\ \ fproj\ \ g)$
proof −
have *1*: $nidx\ \ (NNil\ 0)$
by (*simp add*: *nidx-def*)
have *10*: $nnth\ (NNil\ 0)\ 0 = 0$
by (*simp add*: *nnth-NNil*)
have *11*: *nfinite* $(NNil\ 0)$
by *simp*
have *12*: *nfinite* σ
by (*simp add*: *assms*(*1*) *nlength-eq-enat-nfiniteD zero-enat-def*)
have *2*: $powerinterval\ f\ \sigma\ (NNil\ 0)$
by (*simp add*: *powerinterval-def*)
have *3*: $nlast\ (NNil\ 0) =\ \ nlength\ \sigma$
by (*simp add*: *assms*)
have *4*: $\ g(pfilt\ \sigma\ (NNil\ 0))$
using *assms* **by** *blast*
from *1 10 11 12 2 3 4* **show** *?thesis* **using** *assms*
by (*simp add*: *cppl-fprojection zero-enat-def*)
qed

lemma *cppl-more*:
assumes *nlength* $\sigma > 0$
$(\sigma \models f\ \ fproj\ \ g)$
shows $nlength(cppl\ f\ g\ \sigma) > 0$
using *assms*
by (*metis cppl-fprojection enat-add-sub-same gen-nlength-def gr-zeroI i0-ne-infinity ndropn-nlast ndropn-nlength nlength-NNil nlength-code nnth-nlast the-enat-0 zero-enat-def*)

lemma *cppl-more-infinite*:
assumes *nlength* $\sigma > 0$
$(\sigma \models f\ oproj\ g)$
shows $nlength(cppl\ f\ g\ \sigma) > 0$
using *assms*
by (*simp add*: *cppl-oprojection nfinite-conv-nlength-enat*)

lemma *cppl-more-than-first*:
assumes *nlength* $\sigma > 0$
$(\sigma \models f\ \ fproj\ \ g)$
shows $(nnth\ (cppl\ f\ g\ \sigma)\ 0) = 0$

using *assms*
using *cppl-fprojection* **by** *blast*

lemma *cppl-more-than-first-alt*:
assumes *nlength* $\sigma > 0$
$(\sigma \models f\ oproj\ g)$
shows $(nnth\ (cppl\ f\ g\ \sigma)\ 0) = 0$
using *assms*
using *cppl-oprojection* **by** *blast*

lemma *cppl-more-than-last*:
assumes *nlength* $\sigma > 0$
$(\sigma \models f\ \ fproj\ \ g)$
shows $nlast\ (cppl\ f\ g\ \sigma) = the\text{-}enat(nlength\ \sigma)$
using *assms cppl-fprojection* **by** *blast*

lemma *cppl-sub-more*:
assumes $n < k$
$k \leq nlength\ \sigma$
$((nsubn\ \sigma\ n\ k\ ) \models f\ \ fproj\ \ g)$
shows $nlength(cppl\ f\ g\ (nsubn\ \sigma\ n\ k\ )) > 0$
using *assms cppl-fprojection*[*of f g* $(nsubn\ \sigma\ n\ k\ )$]
using *cppl-more nsubn-nlength-gr-one* **by** *blast*

lemma *cppl-bounds*:
assumes $n < k$
$k \leq nlength\ \sigma$
$((nsubn\ \sigma\ n\ k\ ) \models f\ \ fproj\ \ g)$
$i < nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k\ ))$
shows $0 \leq (nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k))\ i) \wedge (nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k\ ))\ i) \leq k - n$
using *assms*
using *cppl-fprojection*[*of f g* $(nsubn\ \sigma\ n\ k\ )$] **using** *nsubn-nlength*[*of* $\sigma\ n\ k$]
by *simp*
(*metis enat-minus-mono1 idiff-enat-enat min-def nfinite-conv-nlength-enat nidx-less-last-1*
nnth-nlast order-less-imp-le the-enat.simps)

lemma *cppl-nmap-bounds*:
assumes $n < k$
$k \leq nlength\ \sigma$
$((nsubn\ \sigma\ n\ k\ ) \models f\ \ fproj\ \ g)$
$i < nlength\ (nmap\ (\lambda x.\ x+\ \ n)\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k)))$
shows $n \leq (nnth\ (nmap\ (\lambda x.\ x+\ n)\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k)))\ i) \wedge$
$(nnth\ (nmap\ (\lambda x.\ \ x+n)\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k\ )))\ i) \leq k$
using *assms*
using *cppl-bounds* **by** *fastforce*

lemma *cppl-nfirst*:
assumes $(nsubn\ \sigma\ x1a\ (nfirst\ ls)\ ) \models f\ \ fproj\ \ g$
shows $nfirst((nmap\ (\lambda x.\ x+x1a)\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls)\ ))))\ = x1a$
proof −

```
 have 1: ( nidx  (cppl f g (nsubn σ x1a (nfirst ls))) ∧
           nnth (cppl f g (nsubn σ x1a (nfirst ls))) 0 = 0 ∧
           powerinterval f (nsubn σ x1a (nfirst ls)) (cppl f g (nsubn σ x1a (nfirst ls))) ∧
           nfinite (cppl f g (nsubn σ x1a (nfirst ls))) ∧ nfinite (nsubn σ x1a (nfirst ls)) ∧
           nlast (cppl f g (nsubn σ x1a (nfirst ls)))  =
           the-enat(nlength (nsubn σ x1a (nfirst ls))) ∧
          g (pfilt (nsubn σ x1a (nfirst ls)) (cppl f g (nsubn σ x1a (nfirst ls) ))) )
         using cppl-fprojection assms by auto
 have 3:  (nnth (cppl f g (nsubn σ x1a (nfirst ls) )) 0) =0
  by (simp add: 1)
 show ?thesis
 by (metis 3 add-cancel-right-left nfirst-eq-nnth-zero nnth-nmap zero-enat-def zero-le)
qed

lemma cppl-nfirst-same:
 assumes (nsubn σ x1a x1a) ⊨ f  fproj  g
 shows   nfirst((nmap (λx. x+ x1a) (cppl f g (nsubn σ x1a x1a))))  = x1a
proof −
 have 1:  nfirst (NNil x1a ) = x1a
   by (metis NNil-eq-ntake-iff nellist.inject(1))
 from 1 cppl-nfirst show ?thesis
   by (metis assms)
qed

lemma cppl-nlast:
 assumes  ((nsubn σ x (nfirst ls) ) ⊨ f  fproj  g)
          x < nfirst ls
          nfirst ls ≤ nlength σ
 shows   nlast((nmap (λy. y+ x) (cppl f g (nsubn σ x (nfirst ls)))))  = nfirst ls
 proof −
  have 01: ( nidx  (cppl f g (nsubn σ x (nfirst ls))) ∧
            nnth (cppl f g (nsubn σ x (nfirst ls))) 0 = 0 ∧
            powerinterval f (nsubn σ x (nfirst ls)) (cppl f g (nsubn σ x (nfirst ls) )) ∧
            nfinite (cppl f g (nsubn σ x (nfirst ls))) ∧ nfinite (nsubn σ x (nfirst ls)) ∧
            nlast (cppl f g (nsubn σ x (nfirst ls))) = the-enat(nlength (nsubn σ x (nfirst ls))) ∧
            g (pfilt (nsubn σ x (nfirst ls)) (cppl f g (nsubn σ x (nfirst ls)))) )
     using cppl-fprojection assms by (simp add: cppl-fprojection)
  have 02: nlast (cppl f g (nsubn σ x (nfirst ls))) =
             the-enat(nlength (nsubn σ x (nfirst ls) ))
        using 01 by auto
  have 04: x < nfirst ls
    using assms by blast
  have 05: nlength (nsubn σ x (nfirst ls)) = (nfirst ls) −x
   using assms by (metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength)
  have 06: nlast((nmap (λy. y+ x) (cppl f g (nsubn σ x (nfirst ls))))) =
             ((nfirst ls) −x)+x
      using 01 05 by auto
  show ?thesis using 04 06 by auto
qed
```

lemma *cppl-nlast-i*:
assumes $((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \models f\ \ fproj\ \ g)$
$(nnth\ ls\ i) < (nnth\ ls\ (Suc\ i))$
$(nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma$
shows $nlast((nmap\ (\lambda x.\ x+\ (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))\ =$
$(nnth\ ls\ (Suc\ i))$
proof −
have *01*: $(\ nidx\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge$
$nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ 0 = 0 \wedge$
$powerinterval\ f\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
$(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge$
$nfinite\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge$
$nfinite\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
$nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$
$the\text{-}enat(nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge$
$g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
$(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))\ )$
using *assms* **by** (*simp add*: *cppl-fprojection*)
have *02*: $nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$
$the\text{-}enat(nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using *01* **by** *auto*
have *04*: $(nnth\ ls\ i) < (nnth\ ls\ (Suc\ i))$
by (*simp add*: *assms*)
have *05*: $nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
by (*metis assms*(*3*) *enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength*)
have *06*: $nlast((nmap\ (\lambda x.\ x+\ (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))) =$
$((nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)) + (nnth\ ls\ i)$
using *01 05* **by** *auto*
show *?thesis* **using** *04 06* **by** *auto*
qed

lcppl lemmas

lemma *lcppl-nnth*:
assumes *nidx ls*
$i < nlength\ ls$
shows $(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i) =$
$(nmap\ (\lambda x\ .x+\ (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ )))$
using *assms*
proof (*induct i arbitrary*: *ls*)
case *0*
then show *?case*
proof (*cases ls*)
case (*NNil x1*)
then show *?thesis*
using *0.prems*(*2*) **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis* **using** *NCons*
proof (*cases is-NNil x22*)

```
    case True
    then show ?thesis using NCons by simp
     (metis is-NNil-def lcppl-code(2) nnth-NNil)
    next
    case False
    then show ?thesis using NCons by simp
     (metis lcppl-code(3) nellist.collapse(2) nnth-0)
    qed
  qed
next
case (Suc i)
then show ?case
  proof (cases ls)
  case (NNil x1)
  then show ?thesis
  using Suc.prems(2) by auto
  next
  case (NCons x21 x22)
  then show ?thesis
    proof (cases is-NNil x22)
    case True
    then show ?thesis
    by (metis NCons Suc.prems(2) enat-0-iff(1) ile0-eq iless-Suc-eq nat.simps(3) nellist.collapse(1)
        nlength-NCons nlength-NNil)
    next
    case False
    then show ?thesis using NCons by simp
        (metis (no-types, lifting) Suc(1) Suc.prems(1) Suc.prems(2) Suc-ile-eq iless-Suc-eq
          lcppl-code(3) nellist.collapse(2)  nellist.sel(5) nidx-expand nlength-NCons nnth-ntl)
    qed
  qed
qed


lemma lcppl-nlength:
 assumes nfinite ls
         nidx  ls
          nlength ls > 0
 shows    nlength(lcppl f g σ ls) = (epred (nlength ls))
using assms
proof (induct ls rule: nfinite-induct)
case (NNil y)
then show ?case by simp
next
case (NCons x ls)
then show ?case
   proof (cases is-NNil ls)
   case True
   then show ?thesis using NCons
   by (metis co.enat.sel(2) lcppl-code(2) nellist.collapse(1) nlength-NCons nlength-NNil)
```

```
  next
  case False
  then show ?thesis using NCons unfolding nidx-expand by simp
   (metis NCons.prems(1) co.enat.exhaust-sel lcppl-code(3) ndropn-0 ndropn-nlast nellist.collapse(2)
    nellist.disc(1) nidx-LCons-1 nidx-expand nlength-NCons the-enat-0)
  qed
qed

lemma lcppl-nfinite:
 assumes nidx  ls
 shows    nfinite ls ⟷ nfinite(lcppl f g σ ls)  (is ?lhs ⟷ ?rhs)
proof
 assume a: ?lhs
 show   ?rhs
  using a assms
   proof (induct ls rule: nfinite-induct)
   case (NNil y)
   then show ?case by simp
   next
   case (NCons x ls)
   then show ?case
      proof (cases is-NNil ls)
      case True
      then show ?thesis using NCons
      by simp
      next
      case False
      then show ?thesis using NCons by simp
       (metis lcppl-code(3) nellist.collapse(2) nfinite-NConsI)
      qed
   qed
next
 assume b: ?rhs
 show ?lhs
 using b assms
  proof (induct zs≡(lcppl f g σ ls) arbitrary: ls rule: nfinite-induct)
  case (NNil y)
  then show ?case
  by (metis is-NNil-imp-nfinite lcppl.disc(2) nellist.disc(1) nfinite-ntl)
  next
  case (NCons x ls1)
  then show ?case
    proof (cases is-NNil ls1)
    case True
    then show ?thesis using NCons
      by (metis is-NNil-imp-nfinite lcppl-code(3) nellist.collapse(2) nellist.disc(2) nellist.sel(5)
          nfinite-ntl)
    next
    case False
    then show ?thesis using NCons unfolding nidx-expand
```

by (*metis NCons.hyps(2) NCons.prems is-NNil-imp-nfinite lcppl-code(3) nellist.collapse(2) nellist.inject(2) nfinite-ntl nidx-LCons-1*)
qed
qed
qed

lemma *lcppl-nlength-alt*:
assumes ¬*nfinite ls*
nidx ls
shows *nlength*(*lcppl f g σ ls*) = (*epred* (*nlength ls*))
proof −
have *1*: ¬ *nfinite* (*lcppl f g σ ls*)
using *assms(1) assms(2) lcppl-nfinite* **by** *blast*
have *2*: *epred* (*nlength ls*) = ∞
using *assms*
by (*metis enat2-cases epred-Infty nlength-eq-enat-nfiniteD*)
have *3*: *nlength* (*lcppl f g σ ls*) = ∞
by (*meson 1 enat2-cases nlength-eq-enat-nfiniteD*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *lcppl-nlength-zero*:
assumes *nidx ls*
nlength ls = *0*
shows *nlength*(*lcppl f g σ ls*) = *0*
using *assms*
by (*metis* (*no-types*, *lifting*) *is-NNil-def lcppl.ctr(1) le-numeral-extra(3) nlength-NNil ntaken-0 ntaken-all zero-enat-def*)

lemma *lcppl-nlast*:
assumes *nidx ls*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
nlength ls >*0*
shows *nlast* (*lcppl f g σ ls*) =
(*nmap* (λ*x*. *x*+ (*nnth ls* (*the-enat* (*epred*(*nlength ls*)))))
(*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat* (*epred* (*nlength ls*))))
(*nnth ls* (*the-enat* (*nlength ls*))))))
proof −
have *1*: *nlast* (*lcppl f g σ ls*) = (*nnth* (*lcppl f g σ ls*) ((*the-enat* (*nlength* (*lcppl f g σ ls*)))))
using *assms lcppl-nfinite nnth-nlast* **by** *blast*
have *2*: *nlast* (*lcppl f g σ ls*) =
(*nmap* (λ*x*. *x*+ (*nnth ls* ((*the-enat* (*nlength* (*lcppl f g σ ls*))))))
(*cppl f g* (*nsubn σ* (*nnth ls* ((*the-enat* (*nlength* (*lcppl f g σ ls*)))))
(*nnth ls* (*Suc* (*the-enat* (*nlength* (*lcppl f g σ ls*)))))
)))

```
    using assms lcppl-nnth[of ls ((the-enat (nlength (lcppl f g σ ls)))) f g σ]
     lcppl-nlength[of ls f g σ]
     by (metis 1 co.enat.exhaust-sel eSuc-enat enat-ord-simps(2) lcppl-nfinite less-add-same-cancel2
        nfinite-nlength-enat plus-1-eq-Suc the-enat.simps zero-less-Suc zero-less-iff-neq-zero)
  have 3: (nmap (λx. x+ (nnth ls (the-enat (nlength (lcppl f g σ ls)))))
            (cppl f g (nsubn σ (nnth ls ((the-enat (nlength (lcppl f g σ ls)))))
                              (nnth ls (Suc (the-enat (nlength (lcppl f g σ ls)))))
                      ))) =
       (nmap (λx. x+ (nnth ls ((the-enat (epred(nlength ls))))))
            (cppl f g (nsubn σ (nnth ls ((the-enat (epred (nlength ls)))))
                              (nnth ls (Suc (the-enat (epred(nlength ls))))) )))
     using assms lcppl-nlength
     by (metis (no-types, lifting) nellist.map-cong0)
  have 4: (Suc (the-enat (epred(nlength ls)))) = (the-enat (nlength ls))
    by (metis assms(2) assms(5) co.enat.exhaust-sel enat.distinct(2) epred-Infty infinity-ne-i0
       nfinite-conv-nlength-enat not-gr-zero the-enat-eSuc)
  show ?thesis
  using 1 2 3 4 by presburger
qed

lemma lcppl-nlast-nlast:
  assumes nidx  ls
          nfinite ls
          nfinite σ
          nlast ls = the-enat(nlength σ)
          ((nsubn σ (nnth ls (the-enat (epred(nlength ls))) ) (nnth ls (the-enat(nlength ls)))) ⊨ f  fproj  g)
           nlength ls > 0
  shows     nlast (nlast (lcppl f g σ ls)) = the-enat(nlength σ)
proof −
  have 2: (nlast (lcppl f g σ ls)) =
            (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls)))))
               (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
                                 (nnth ls (the-enat (nlength ls))) )))
    using assms lcppl-nlast[of ls σ f g]  by blast
  have 3: nlast (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls)))))
                (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
                                  (nnth ls (the-enat (nlength ls))) ))) =
          (λx. x+ (nnth ls (the-enat (epred(nlength ls)))))
          (nlast  (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
                                    (nnth ls (the-enat (nlength ls))) )))
       using nnth-nmap assms by (simp add: cppl-fprojection)
  have 4: nnth ls (the-enat (epred (nlength ls))) < nnth ls (Suc (the-enat (epred (nlength ls))))
    using assms unfolding nidx-expand
    by (metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat enat.simps(3) enat-ord-simps(2) enat-the-enat
       i0-less i0-ne-infinity lessI nfinite-conv-nlength-enat)
  have 5: enat (nnth ls (Suc (the-enat (epred (nlength ls))))) ≤ nlength σ
    using assms
    by (metis Orderings.order-eq-iff co.enat.exhaust-sel enat.simps(3) enat-the-enat epred-simps(5)
        i0-less i0-ne-infinity nfinite-conv-nlength-enat nnth-nlast the-enat-eSuc)
  have 6: nlast (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls))))))
```

```
                (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
                                   (nnth ls (the-enat (nlength ls))) ))) =
          nnth ls (Suc (the-enat (epred (nlength ls))))
   using cppl-nlast-i[of f g σ ls (the-enat (epred (nlength ls)))] assms 4 5
   by (metis  co.enat.exhaust-sel enat.simps(3) epred-simps(5) i0-less i0-ne-infinity
      nfinite-conv-nlength-enat the-enat-eSuc)
 have 7: nnth ls (Suc (the-enat (epred (nlength ls)))) = the-enat(nlength σ)
   using assms
   by (metis  co.enat.collapse enat-eSuc-iff nfinite-nlength-enat nnth-nlast the-enat.simps
     zero-less-iff-neq-zero)
 show ?thesis
 by (simp add: 2 6 7)
qed

lemma lcppl-zero-zero:
  assumes nidx  ls
          nfinite ls
          nfinite σ
           nlast ls  = the-enat(nlength σ)
           nlength ls = 0
 shows   (nnth (lcppl f g σ ls) 0) =
         (nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0))))
proof (cases ls)
case (NNil x1)
then show ?thesis
by (metis lcppl-code(1)  nnth-NNil)
next
case (NCons x21 ls)
then show ?thesis
using assms(5) by auto
qed

lemma lcppl-nfirst:
 assumes nidx ls
         nfinite ls
         nfinite σ
         nlast ls = the-enat(nlength σ)
         (∀ i.  i<nlength ls ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f  fproj  g))
          nlength ls > 0
 shows    nfirst(nfirst((lcppl f g σ ls))) = nfirst ls
proof −
 have 01: (nfirst((lcppl f g σ ls))) =
          (nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls (Suc 0)))))
     using assms
     by (metis lcppl-nnth ndropn-0 ndropn-nfirst zero-enat-def)
 have 02: nlength ls > 0 ⟶
          nfirst((nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls (Suc 0)))))) =
           (nnth ls 0)
           using assms 01 cppl-fprojection[of f g  ]
           by (metis cppl-nfirst enat-0-iff(1) ndropn-nfirst)
```

show *?thesis*
using *01 02 assms*
by (*metis ndropn-0 ndropn-nfirst*)
qed

lemma *lcppl-nfirst-alt*:
assumes *nidx ls*
¬*nfinite ls*
¬*nfinite σ*
(∀ *i*. *i*<*nlength ls* ⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*))
shows *nfirst*(*nfirst*((*lcppl f g σ ls*))) = *nfirst ls*
proof −
have *01*: (*nfirst*((*lcppl f g σ ls*))) =
(*nmap* (λ*x*. *x*+ (*nnth ls 0*)) (*cppl f g* (*nsubn σ* (*nnth ls 0*) (*nnth ls* (*Suc 0*)))))
using *assms*
by (*metis i0-less lcppl-nnth nfinite-ntaken nlength-eq-enat-nfiniteD nnth-NNil nnth-nlast ntaken-0 ntaken-nlast zero-enat-def*)
have *02*: *nfirst*((*nmap* (λ*x*. *x*+ (*nnth ls 0*)) (*cppl f g* (*nsubn σ* (*nnth ls 0*) (*nnth ls* (*Suc 0*)))))) =
(*nnth ls 0*)
using *assms 01 cppl-fprojection*[*of f g*]
by (*metis* (*no-types*, *lifting*) *add-0 gr-zeroI ndropn-0 ndropn-nfirst nlength-eq-enat-nfiniteD nnth-nmap zero-enat-def zero-le*)
show *?thesis*
using *01 02 assms*
by (*metis ndropn-0 ndropn-nfirst*)
qed

lemma *lcppl-nfusecat-nlastnfirst*:
assumes *nfinite ls*
nidx ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
((*nlength ls* = *0* ∧ ((*nsubn σ* (*nnth ls 0*) (*nnth ls 0*)) ⊨ *f fproj g*)) ∨
(*nlength ls* > *0* ∧ (∀ *i*. *i*<*nlength ls* ⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*))))
shows *nlastnfirst* (*lcppl f g σ ls*)
proof (*cases is-NNil ls*)
case *True*
then show *?thesis*
by (*metis is-NNil-def lcppl-code*(*1*) *nlastnfirst-NNil*)
next
case *False*
then show *?thesis*
proof (*auto simp add*: *nlastnfirst-def1*)
fix *i*
assume *a*: *enat* (*Suc i*) ≤ *nlength* (*lcppl f g σ ls*)
assume *b*: ¬ *is-NNil ls*
show *nlast* (*nnth* (*lcppl f g σ ls*) *i*) = *nfirst* (*nnth* (*lcppl f g σ ls*) (*Suc i*))
proof −
have *1*: *nlength ls* > *0*

by (*metis a assms*(*2*) *assms*(*5*) *enat.inject ile0-eq lcppl-nlength-zero old.nat.distinct*(*1*) *zero-enat-def*)
have *2*: (∀ *i*. *i*<*nlength ls*⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*))
using *assms*(*5*) **by** *auto*
have *3*: *nidx ls*
by (*simp add*: *assms*(*2*))
have *4*: *i*< *nlength ls*
by (*metis 1 3 Suc-ile-eq a assms*(*1*) *co.enat.exhaust-sel dual-order.order-iff-strict iless-Suc-eq lcppl-nlength less-numeral-extra*(*3*))
have *5*: (*nnth* (*lcppl f g σ ls*) *i*) =
nmap (λ*x*::*nat*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))
using *lcppl-nnth*[*of ls i f g σ*]
using *3 4* **by** *blast*
have *6*: (*Suc i*) < *nlength ls*
by (*metis 1 3 a assms*(*1*) *co.enat.exhaust-sel iless-Suc-eq lcppl-nlength less-numeral-extra*(*3*))
have *7*: (*nnth* (*lcppl f g σ ls*) (*Suc i*)) =
nmap (λ*x*::*nat*. *x* + *nnth ls* (*Suc i*)) (*cppl f g* (*nsubn σ* (*nnth ls* (*Suc i*)) (*nnth ls* (*Suc* (*Suc i*)))))
using *lcppl-nnth*[*of ls Suc i f g σ*]
using *3 6* **by** *blast*
have *71*: *nnth ls i* < *nnth ls* (*Suc i*)
using *assms nidx-expand*[*of ls*] *6 order-less-imp-le* **by** *blast*
have *72*: *enat* (*nnth ls* (*Suc i*)) ≤ *nlength σ*
using *assms*
by (*metis 3 6 dual-order.order-iff-strict enat-ord-simps*(*1*) *nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps*)
have *8*: *nlast* (*nmap* (λ*x*::*nat*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
(*nnth ls* (*Suc i*))
using *assms 2 3 4 5 6 cppl-nlast-i*[*of f g σ ls i*]
using *71 72* **by** *blast*
have *9*: *nfirst* (*nmap* (λ*x*::*nat*. *x* + *nnth ls* (*Suc i*))
(*cppl f g* (*nsubn σ* (*nnth ls* (*Suc i*)) (*nnth ls* (*Suc* (*Suc i*)))))) =
(*nnth ls* (*Suc i*))
using *2 6 cppl-fprojection*[*of f g*]
by (*metis add-0 nlast-NNil nnth-nmap ntaken-0 ntaken-nlast zero-enat-def zero-order*(*1*))
show *?thesis* **using** *8 9 5 7* **by** *presburger*
qed
qed
qed

lemma *lcppl-nfusecat-nlastnfirst-alt*:
assumes ¬*nfinite ls*
nidx ls
¬*nfinite σ*
(∀ *i*. *i*<*nlength ls*⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*))
shows *nlastnfirst* (*lcppl f g σ ls*)
proof (*auto simp add*: *nlastnfirst-def1*)
fix *i*
assume *a*: *enat* (*Suc i*) ≤ *nlength* (*lcppl f g σ ls*)
show *nlast* (*nnth* (*lcppl f g σ ls*) *i*) = *nfirst* (*nnth* (*lcppl f g σ ls*) (*Suc i*))
proof −

```
  have 2: (∀ i.  i<nlength ls⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g))
    using assms by auto
  have 3: nidx ls
    by (simp add: assms)
  have 4:  i< nlength ls
    by (meson assms(1) enat-iless linorder-less-linear nfinite-conv-nlength-enat)
  have 5: (nnth (lcppl f g σ ls) i) =
          nmap (λx::nat. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
    using lcppl-nnth[of ls i f g σ]
    using 3 4 by blast
  have 6:  (Suc i) < nlength ls
    by (metis 4 assms(1) eSuc-enat ileI1 nlength-eq-enat-nfiniteD order-neq-le-trans)
  have 7: (nnth (lcppl f g σ ls) (Suc i)) =
          nmap (λx::nat. x + nnth ls (Suc i))
               (cppl f g (nsubn σ (nnth ls (Suc i)) (nnth ls (Suc (Suc i)))))
    using lcppl-nnth[of ls Suc i  f g σ]
      using 3 6 by blast
  have 8: nlast (nmap (λx::nat. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) =
          (nnth ls (Suc i))
    using assms 2 3 4 5 6
    by (simp add: cppl-nlast-i nfinite-conv-nlength-enat nidx-expand)
  have 9: nfirst (nmap (λx::nat. x + nnth ls (Suc i))
                       (cppl f g (nsubn σ (nnth ls (Suc i)) (nnth ls (Suc (Suc i)))))) =
          (nnth ls (Suc i))
    using 2 6 cppl-fprojection[of f g ]
    by (metis add-0 nlast-NNil nnth-nmap ntaken-0 ntaken-nlast zero-enat-def zero-order(1))
  show ?thesis using 8 9 5 7 by presburger
 qed
qed

lemma lcppl-nfusecat-nlength:
 assumes nidx  ls
         nfinite ls
         nfinite σ
         nlast ls  = the-enat(nlength σ)
         ((nlength ls =0 ∧ ((nsubn σ (nnth ls 0 ) (nnth ls 0 )  ⊨ f  fproj  g))) ∨
           (nlength ls >0 ∧ (∀  i. i<nlength ls ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f  fproj  g))))
 shows   (nlength ls =0 ⟶ nlength(nfusecat (lcppl f g σ ls)) = 0) ∧
         (nlength ls >0 ⟶
           nlength(nfusecat (lcppl f g σ ls)) =
           ( (∑ k=0..(the-enat (epred(nlength ls))). ( (nlength (nnth (lcppl f g σ ls) k))))))
proof −
 have 1: nlastnfirst (lcppl f g σ ls)
  using assms lcppl-nfusecat-nlastnfirst by blast
 have 20: nlength ls = 0 ⟶
           (lcppl f g σ ls) =
            (NNil ((nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0) )))))
     by (metis ile0-eq lcppl-code(1)  nellist-eq-nnth-eq nlength-NNil nnth-NNil  the-enat.simps
         the-enat-0)
 have 2: nlength ls = 0 ⟶
```

```
        nfusecat (lcppl f g σ ls) =
        ((nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0) ))))
      using 20 nfusecat-NNil by auto
 have 3: nlength ls = 0 ⟶
        nlength ((nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0) )))) = 0
   using assms
   by (metis cppl-empty diff-self-eq-0 idiff-enat-enat less-numeral-extra(3) min.orderE ndropn-nlast
       ndropn-nlength nlength-NNil nlength-nmap nnth-nlast not-le-imp-less nsubn-nlength the-enat-0
       zero-enat-def)
 have 4: nlength ls = 0 ⟶ nlength(nfusecat (lcppl f g σ ls)) = 0
   using 2 3 by auto
 have 5: nfinite (lcppl f g σ ls)
   using assms(1) assms(2) lcppl-nfinite by blast
 have 6:  all-nfinite (lcppl f g σ ls)
 by (metis (no-types, lifting) 20 all-nfinite-nnth-a assms(1) assms(2) assms(5) co.enat.exhaust-sel
     cppl-fprojection iless-Suc-eq lcppl-nlength lcppl-nnth nfinite-nmap nnth-NNil
     not-less-iff-gr-or-eq)
 have 7: nlength (lcppl f g σ ls) = (epred(nlength ls))
   by (metis assms(1) assms(2) assms(5) epred-0 lcppl-nlength lcppl-nlength-zero)
 have 8: nlength ls > 0 ⟶
          nlength(nfusecat (lcppl f g σ ls)) =
          (∑ k=0..(the-enat (epred(nlength ls))). ((nlength (nnth (lcppl f g σ ls) k))))
  using nfusecat-nlength-nfinite[of (lcppl f g σ ls)] 5 6 7 1 by presburger
 show ?thesis
 using 4 8 by blast
qed

lemma lcppl-nfusecat-nidx:
 assumes nidx  ls
        nnth ls 0 = 0
        nfinite ls
        nfinite σ
         nlast ls  = the-enat(nlength σ)
        (∀  i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g)
        nlength ls > 0
 shows   nidx  (nfusecat ( (lcppl f g σ ls)))
proof −
 have 0: nlength σ >0 ⟶ nlength ls >0
   using assms by auto
 have 2: nlength σ >0 ⟶ nlastnfirst (lcppl f g σ ls)
   using assms lcppl-nfusecat-nlastnfirst by blast
 have 3:  (∀ i<nlength ls. nidx (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))
    using assms cppl-fprojection by auto
 have 4:  (∀ i<nlength ls.
             nidx (nmap (λx. x+(nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))))
   using 3 by (simp add: Suc-ile-eq nidx-expand)
 have 5: nlength σ >0 ⟶
         (∀ i<nlength ls. nidx (nnth (lcppl f g σ ls) i) )
   using assms by (simp add: 4 lcppl-nnth)
 have 6: nlength σ >0 ⟶ nlength (lcppl f g σ ls) = epred(nlength ls)
```

using *assms lcppl-nlength* **by** *blast*
have *7*: *nlength* $\sigma > 0 \longrightarrow$
$(\forall\, i \leq$*nlength ls*.
nidx (*nnth* (*lcppl f g* σ *ls*) *i*))
using *5 assms*
by (*metis 6 Extended-Nat.eSuc-mono antisym-conv2 co.enat.exhaust-sel enat-the-enat epred-simps*(*5*) *i0-ne-infinity iless-Suc-eq lcppl-nfinite nnth-beyond nnth-nlast*)
have *68*: $\bigwedge i.$ *i*<*nlength ls* $\Longrightarrow$ *0* < *nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\Longrightarrow$
(*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ (*f fproj g*) $\Longrightarrow$
0 < *nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
using *cppl-more* **by** *blast*
have *69*: $\bigwedge i.$ *i*<*nlength ls* $\Longrightarrow$ *0* < *nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
using *assms* **unfolding** *nidx-expand* **by** *simp*
(*metis assms*(*1*) *dual-order.order-iff-strict eSuc-enat enat-ord-simps*(*2*) *ileI1 less-numeral-extra*(*3*) *nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast nsubn-nlength-gr-one the-enat.simps*)
have *70*: *nlength* $\sigma > 0 \longrightarrow (\forall\, i <$*nlength ls*. (*0*::*enat*) < *nlength* (*nnth* (*lcppl f g* σ *ls*) *i*))
using *lcppl-nnth*[*of ls - f g* σ] *68 69 assms*(*1*) *assms*(*6*) **by** *auto*
have *700*: *nlength* $\sigma > 0 \longrightarrow (\forall\, lx \in$*nset* (*lcppl f g* σ *ls*). (*0*::*enat*) < *nlength lx*)
using *70 assms*
by (*metis co.enat.exhaust-sel iless-Suc-eq in-nset-conv-nnth lcppl-nlength less-numeral-extra*(*3*))
have *71*: *nlength* $\sigma > 0 \longrightarrow (\forall\, lx \in$ *nset* (*lcppl f g* σ *ls*). *nfinite lx*)
using *assms* **unfolding** *cppl-fprojection*
by (*metis* (*no-types*, *lifting*) *co.enat.exhaust-sel i0-less iless-Suc-eq in-nset-conv-nnth lcppl-nlength lcppl-nnth nfinite-nmap*)
have *8*: *nlength* $\sigma > 0 \longrightarrow$
nidx (*nfusecat* ((*lcppl f g* σ *ls*)))
using *assms nidx-nfusecat*[*of* ((*lcppl f g* σ *ls*))] *700 71*
by (*metis 2 5 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength zero-less-iff-neq-zero*)
from *8* **show** *?thesis* **using** *assms*
by (*metis gr-zeroI less-numeral-extra*(*3*) *nfinite-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps zero-enat-def*)
qed

lemma *lcppl-nfusecat-nidx-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
$\neg$*nfinite ls*
$\neg$*nfinite* σ
(∀ *i*<*nlength ls*. (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f fproj g*)
shows *nidx* (*nfusecat* ((*lcppl f g* σ *ls*)))
proof −
have *2*: *nlastnfirst* (*lcppl f g* σ *ls*)
using *assms lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *3*: (∀ *i*<*nlength ls*. *nidx* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))))
using *assms cppl-fprojection* **by** *auto*
have *4*: (∀ *i*<*nlength ls*.
nidx (*nmap* (λ*x*. *x*+(*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *3* **by** (*simp add*: *Suc-ile-eq nidx-expand*)
have *5*: (∀ *i*<*nlength ls*. *nidx* (*nnth* (*lcppl f g* σ *ls*) *i*))
using *assms* **by** (*simp add*: *4 lcppl-nnth*)

have *7*: (∀ *i*<*nlength ls*.
nidx (*nnth* (*lcppl f g σ ls*) *i*))
using *5* **by** *simp*
have *60*: ⋀*j*. (*nsubn σ* (*nnth ls j*) (*nnth ls* (*Suc j*))) ⊨ (*f fproj g*)
by (*metis assms*(*3*) *assms*(*5*) *leI nfinite-ntaken ntaken-all*)
have *61*: ⋀*j*. *nnth ls j* < *nnth ls* (*Suc j*)
by (*metis Suc-ile-eq assms*(*1*) *assms*(*3*) *nfinite-ntaken nidx-expand not-le-imp-less ntaken-all*)
have *62*: ⋀*j*. *enat* (*nnth ls* (*Suc j*)) ≤ *nlength σ*
by (*meson assms*(*4*) *enat-iless enat-less-imp-le nfinite-conv-nlength-enat*)
have *63*: ⋀*j*. *nlast* (*nmap* (λ*x*::*nat*. *x* + *nnth ls j*) (*cppl f g* (*nsubn σ* (*nnth ls j*) (*nnth ls* (*Suc j*))))) =
nnth ls (*Suc j*)
using *60 61 62 cppl-nlast-i* **by** *blast*
have *68*: ⋀*i*. *i*<*nlength ls* ⟹ *0* < *nlength* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⟹
(*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ (*f fproj g*) ⟹
0 < *nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))
using *cppl-more* **by** *blast*
have *69*: ⋀*i*. *i*<*nlength ls* ⟹ *0* < *nlength* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
using *assms* **unfolding** *nidx-expand* **using** *61 62 nsubn-nlength-gr-one* **by** *blast*
have *700*: *nlength σ* >*0* ⟶ (∀ *i*<*nlength ls*. (*0*::*enat*) < *nlength* (*nnth* (*lcppl f g σ ls*) *i*))
using *lcppl-nnth*[*of ls - f g σ*] *68 69 assms* **by** (*simp*)
have *701*: *nlength σ* >*0* ⟶ (∀ *lx*∈*nset* (*lcppl f g σ ls*). (*0*::*enat*) < *nlength lx*)
using *700 assms*
by (*metis co.enat.exhaust-sel iless-Suc-eq in-nset-conv-nnth lcppl-nlength-alt*
nlength-eq-enat-nfiniteD zero-enat-def)
have *70*: (∀ *lx*∈*nset* (*lcppl f g σ ls*). (*0*::*enat*) < *nlength lx*)
using *701 assms nfinite-conv-nlength-enat zero-enat-def* **by** *fastforce*
have *71*: (∀ *lx* ∈ *nset* (*lcppl f g σ ls*). *nfinite lx*)
using *assms*
by (*metis* (*no-types*, *lifting*) *co.enat.exhaust-sel cppl-fprojection iless-Suc-eq in-nset-conv-nnth*
lcppl-nlength-alt lcppl-nnth nfinite-nmap nlength-eq-enat-nfiniteD zero-enat-def)
have *8*: *nidx* (*nfusecat* ((*lcppl f g σ ls*)))
using *assms nidx-nfusecat*[*of* ((*lcppl f g σ ls*))] *70 71*
by (*metis 2 7 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength-alt*
nlength-eq-enat-nfiniteD zero-enat-def)
from *8* **show** *?thesis* **using** *assms* **by** *blast*
qed

lemma *lcppl-nlength-all-nfinite*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ >*0*
shows (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nfinite*(*nnth* (*lcppl f g σ ls*) *j*))
proof
fix *j*
show *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶ *nfinite* (*nnth* (*lcppl f g σ ls*) *j*)
proof −

have *1*: *nlength σ>0 ⟶ nlastnfirst (lcppl f g σ ls)*
using *assms*
by (*metis enat.distinct(2) enat-the-enat i0-less lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *2*: *nlength σ>0 ⟶ nlength ls >0*
using *assms*
by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *3*: *nlength σ>0 ⟶ j ≤ nlength (lcppl f g σ ls) ⟶*
(nnth (lcppl f g σ ls) j) =
(nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)))))
using *assms lcppl-nnth*[*of ls j f g σ*]
by (*metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero*)
have *4*: *nlength σ>0 ⟶ j ≤ nlength (lcppl f g σ ls) ⟶*
nfinite (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)))))
using *assms 2*
by (*metis co.enat.exhaust-sel cppl-fprojection iless-Suc-eq lcppl-nlength nfinite-nmap not-gr-zero*)
show *?thesis*
using *3 4 assms(7)* **by** *presburger*
qed
qed

lemma *lcppl-nlength-all-gr-zero*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
nlength σ >0
shows *(∀ j ≤ nlength (lcppl f g σ ls). nlength(nnth (lcppl f g σ ls) j) > 0)*
proof
fix *j*
show *j ≤ nlength (lcppl f g σ ls) ⟶ 0 < nlength (nnth (lcppl f g σ ls) j)*
proof −
have *1*: *nlength σ>0 ⟶ nlastnfirst (lcppl f g σ ls)*
using *assms*
by (*metis enat.distinct(2) enat-the-enat i0-less lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *2*: *nlength σ>0 ⟶ nlength ls >0*
using *assms*
by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *3*: *nlength σ>0 ⟶ j ≤ nlength (lcppl f g σ ls) ⟶*
(nnth (lcppl f g σ ls) j) =
(nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)))))
using *assms lcppl-nnth*[*of ls j f g σ*]
by (*metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero*)
have *4*: *nlength σ>0 ⟶ j ≤ nlength (lcppl f g σ ls) ⟶*
nlength (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j))))) > 0
using *assms 2 cppl-more*[*of (nsubn σ (nnth ls j) (nnth ls (Suc j))) f g*]
by *simp*

(*metis co.enat.exhaust-sel eSuc-enat enat-le-plus-same(2) enat-ord-simps(1) gen-nlength-def i0-less ileI1 iless-Suc-eq lcppl-nlength nfinite-nlength-enat nidx-expand nidx-less-eq nlength-code nnth-nlast nsubn-nlength-gr-one the-enat.simps*)
show *?thesis*
using *3 4 assms*(7) **by** *presburger*
qed
qed

lemma *lcppl-nlength-all-nfinite-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nfinite*(*nnth* (*lcppl f g σ ls*) *j*))
proof
fix *j*
show *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶ *nfinite* (*nnth* (*lcppl f g σ ls*) *j*)
proof −
have *1*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *3*: *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶
(*nnth* (*lcppl f g σ ls*) *j*) =
(*nmap* (λ*x*. *x*+ (*nnth ls j*)) (*cppl f g* (*nsubn σ* (*nnth ls j*) (*nnth ls* (*Suc j*)))))
using *assms lcppl-nnth*[*of ls j f g σ*] **by** (*metis leI nfinite-ntaken ntaken-all*)
have *4*: *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶
nfinite (*nmap* (λ*x*. *x*+ (*nnth ls j*)) (*cppl f g* (*nsubn σ* (*nnth ls j*) (*nnth ls* (*Suc j*)))))
using *assms*
by (*metis Suc-ile-eq cppl-fprojection linorder-le-cases nfinite-nmap nfinite-ntaken ntaken-all*)
show *?thesis*
using *3 4 assms* **by** *presburger*
qed
qed

lemma *lcppl-nlength-all-gr-zero-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)
proof
fix *j*
show *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶ *0* < *nlength* (*nnth* (*lcppl f g σ ls*) *j*)
proof −
have *1*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *3*: *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶

```
        (nnth (lcppl f g σ ls) j) =
          (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)))))
    using assms lcppl-nnth[of ls j f g σ ] by (metis leI nfinite-ntaken ntaken-all)
  have 4:  j ≤ nlength (lcppl f g σ ls) ⟶
         nlength (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j))))) > 0
    using assms
    by (metis Suc-ile-eq cppl-more enat-ile linorder-le-cases nfinite-conv-nlength-enat
       nidx-expand nlength-nmap nsubn-nlength-gr-one)
  show ?thesis
  using  3 4 assms by presburger
  qed
qed

lemma lcppl-nlast-nnth:
 assumes nidx  ls
        nnth ls 0 = 0
        nfinite ls
        nfinite σ
        nlast ls  = the-enat(nlength σ)
        (∀  i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g)
         nlength σ > 0
         j ≤ nlength (lcppl f g σ ls)
 shows  nlast(nnth (lcppl f g σ ls) j) = (nnth ls (Suc j))
proof −
 have 0: nlength σ >0 ⟶ nlength ls >0
    using assms
    by (metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0
       zero-less-iff-neq-zero)
 have 1: nlength σ>0 ⟶
         j ≤ nlength (lcppl f g σ ls)   ⟶
          nnth (lcppl f g σ ls) j =
           (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)) )))
    using assms lcppl-nnth[of ls j f g σ]
       by (metis 0 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero)
 have 2: nlength σ>0 ⟶
         j ≤ nlength (lcppl f g σ ls)   ⟶ (nnth ls (Suc j)) ≤ nlength σ
     using assms
    by (metis dual-order.eq-iff enat.simps(3) enat-ord-simps(1) enat-the-enat
       nfinite-conv-nlength-enat nidx-all-le-nlast nnth-beyond not-le-imp-less)
 have 3: nlength σ>0 ⟶
         j ≤ nlength (lcppl f g σ ls)   ⟶
         nlast (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j))))) =
          (nnth ls (Suc j))
    using assms cppl-nlast-i[of f g σ ls j]
     by (metis 0 2 Suc-ile-eq co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength nidx-expand)
 show ?thesis using 1 3 assms
 by presburger
qed

lemma lcppl-nlast-nnth-alt:
```

assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
j ≤ nlength (lcppl f g σ ls)
shows *nlast(nnth (lcppl f g σ ls) j) = (nnth ls (Suc j))*
proof −
have *1*:
j ≤ nlength (lcppl f g σ ls) ⟶
nnth (lcppl f g σ ls) j =
(nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)))))

using *assms lcppl-nnth[of ls j f g σ]* **by** *(metis leI nfinite-ntaken ntaken-all)*
have *2*:
j ≤ nlength (lcppl f g σ ls) ⟶ (nnth ls (Suc j)) ≤ nlength σ
using *assms*
by *(simp add: nfinite-conv-nlength-enat)*
have *3*: *j ≤ nlength (lcppl f g σ ls) ⟶*
nlast (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j))))) =
(nnth ls (Suc j))
using *assms cppl-nlast-i[of f g σ ls j]*
by *(metis 2 co.enat.exhaust-sel eSuc-enat ileI1 iless-Suc-eq lcppl-nlength-alt*
nidx-expand nlength-eq-enat-nfiniteD zero-enat-def)
show *?thesis* **using** *1 3 assms*
by *presburger*
qed

lemma *lcppl-nfusecat-pfilt-fpower-help*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
nlength σ > 0
shows *(∀ i<nlength ls. g (pfilt σ (nnth (lcppl f g σ ls) i)))*
proof −
have *1*: *(∀ i<nlength ls. g (pfilt (nsubn σ (nnth ls i) (nnth ls (Suc i)))*
(cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))))
using *assms cppl-fprojection* **by** *blast*
have *2*: *nlength σ >0 ⟶*
(∀ i<nlength ls.
(pfilt σ (nnth (lcppl f g σ ls) i)) =
(pfilt σ (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))))))

using *assms* **by** *(simp add: lcppl-nnth)*
have *3*: *nlength σ >0 ⟶*
(∀ i<nlength ls.
nlength (pfilt σ (nnth (lcppl f g σ ls) i)) =

```
        nlength (pfilt (nsubn σ (nnth ls i) (nnth ls (Suc i)))
                       (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) )
      )

    by (simp add: 2 pfilt-nlength)
have 4: nlength σ >0 ⟶
      (∀ i<nlength ls.
       (∀ j≤ nlength (pfilt σ (nnth (lcppl f g σ ls) i)).
         (nnth (pfilt σ (nmap (λx. x+ (nnth ls i))
                        (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))) j)
          =
          (nnth σ ((nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) j) +(nnth ls i) ))
       )
      )
    using nnth-nmap  pfilt-nmap by (metis 2 nlength-nmap)
have 5: nlength σ >0 ⟶
      (∀ i<nlength ls.
       (∀ j≤ nlength (pfilt σ (nnth (lcppl f g σ ls) i)).
         (nnth (pfilt (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
                      (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) ) j) =
         (nnth (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
             (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) j))
       ))
      by (simp add: 3 pfilt-nnth)
have 6: nlength σ >0 ⟶
      (∀ i<nlength ls.
         (nnth ls (Suc i)) ≤ nlength σ
      )
   using assms
   by (metis eSuc-enat enat-le-plus-same(2) enat-ord-simps(1) gen-nlength-def ileI1
       nfinite-conv-nlength-enat nidx-less-eq nlength-code nnth-nlast the-enat.simps)
have 7: nlength σ >0 ⟶
      (∀ i<nlength ls.
          (nnth ls i) ≤ (nnth ls (Suc i))
      )
    using assms
    by (metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)
have 70: ⋀i j . i<nlength ls ⟹ (nnth ls i) < (nnth ls (Suc i)) ⟹
  enat (nnth ls (Suc i)) ≤ nlength σ ⟹
  (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f  fproj  g)  ⟹
  enat j < nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ⟹
  0 ≤ nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) j ∧
  nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) j ≤ (nnth ls (Suc i)) − (nnth ls i)
      using cppl-bounds by blast
have 71: ⋀i.  i<nlength ls ⟹ (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f  fproj  g)  ⟹
          nnth ls i < nnth ls (Suc i) ⟹
          enat (nnth ls (Suc i)) ≤ nlength σ ⟹
          nlast (nmap (λx. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) =
          nnth ls (Suc i)
    using cppl-nlast-i by blast
```

have *8*: *nlength* σ *>0* $\longrightarrow$
(∀ *i<nlength ls.*
(∀ *j≤ nlength* (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)).
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*) ≤
(*nnth ls* (*Suc i*)) − (*nnth ls i*)
))
using *6 assms 70 71* **unfolding** *cppl-fprojection nidx-expand* **by** *simp*
(*metis add-diff-cancel-right′ dual-order.eq-iff eSuc-enat ileI1 not-le-imp-less ntaken-all ntaken-nlast*)
have *9*: *nlength* σ *>0* $\longrightarrow$
(∀ *i<nlength ls.*
(∀ *j≤ nlength* (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)).
(*nnth* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*)) =
(*nnth* σ ((*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*) + (*nnth ls i*)))
))
using *6 7 8*
by (*simp add*: *add.commute nsubn-def1 ntaken-nnth*)
have *10*: *nlength* σ *>0* $\longrightarrow$
(∀ *i<nlength ls.*
(∀ *j≤ nlength* (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)).
(*pfilt* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
(*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*))
))
using *2 3 4 9* **by** (*simp add*: *pfilt-expand pfilt-nlength pfilt-nnth*)
show *?thesis* **using** *1 10 assms* **by** *fastforce*
qed

lemma *lcppl-nfusecat-pfilt-fpower-help-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬*nfinite ls*
¬*nfinite* σ
(∀ *i<nlength ls.* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows (∀ *i<nlength ls. g* (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)))
proof −
have *1*: (∀ *i<nlength ls. g* (*pfilt* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *assms cppl-fprojection* **by** *blast*
have *2*: (∀ *i<nlength ls.*
(*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)) =
(*pfilt* σ (*nmap* (λ*x. x+* (*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))))))
using *assms* **by** (*simp add*: *lcppl-nnth*)
have *3*: (∀ *i<nlength ls.*
nlength (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)) =
nlength (*pfilt* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))))
)
by (*simp add*: *2 pfilt-nlength*)

have *4*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).
(*nnth* (*pfilt σ* (*nmap* (λ*x*. *x*+ (*nnth ls i*))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))) *j*)
=
(*nnth σ* ((*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*) +(*nnth ls i*)))
)
)
using *nnth-nmap pfilt-nmap* **by** (*metis 2 nlength-nmap*)
have *5*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).
(*nnth* (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) *j*) =
(*nnth* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*))
))
by (*simp add*: *3 pfilt-nnth*)
have *6*: (∀ *i*<*nlength ls*.
(*nnth ls* (*Suc i*)) ≤ *nlength σ*
)
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *7*: (∀ *i*<*nlength ls*.
(*nnth ls i*) ≤ (*nnth ls* (*Suc i*))
)
using *assms*
by (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *70*: ⋀*i j* . *i*<*nlength ls* ⟹ (*nnth ls i*) < (*nnth ls* (*Suc i*)) ⟹
enat (*nnth ls* (*Suc i*)) ≤ *nlength σ* ⟹
(*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ (*f fproj g*) ⟹
enat j < *nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) ⟹
0 ≤ *nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j* ∧
nnth (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j* ≤ (*nnth ls* (*Suc i*)) − (*nnth ls i*)
using *cppl-bounds* **by** *blast*
have *71*: ⋀*i*. *i*<*nlength ls* ⟹ (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ (*f fproj g*) ⟹
nnth ls i < *nnth ls* (*Suc i*) ⟹
enat (*nnth ls* (*Suc i*)) ≤ *nlength σ* ⟹
nlast (*nmap* (λ*x*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
nnth ls (*Suc i*)
using *cppl-nlast-i* **by** *blast*
have *8*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*) ≤
(*nnth ls* (*Suc i*)) − (*nnth ls i*)
))
using *assms* **using** *6 assms 70 71* **unfolding** *cppl-fprojection nidx-expand* **by** *simp*
(*metis add-diff-cancel-right' dual-order.eq-iff eSuc-enat ileI1 not-le-imp-less ntaken-all*
ntaken-nlast)
have *9*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).

(*nnth* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*)) =
(*nnth* σ ((*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*) + (*nnth ls i*)))
))
using *6 7 8* **by** (*simp add*: *add.commute nsubn-def1 ntaken-nnth*)
have *10*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength* (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)).
(*pfilt* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
(*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*))
))
using *2 3 4 9* **by** (*simp add*: *pfilt-expand pfilt-nlength pfilt-nnth*)
show *?thesis* **using** *1 10 assms* **by** *fastforce*
qed

lsum lemmas

lemma *pfilt-nfuse-lsum-a*:
assumes *nlast nell* = *nfirst nell1*
nlength nell > *0*
nlength nell1 > *0*
nfinite nell
nfinite nell1
shows *pfilt* (*nfuse nell nell1*) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*))) =
pfilt nell1 (*lsum* (*NNil nell1*) *0*)
proof −
have *1*: (*pfilt* (*nfuse nell nell1*) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))) =
nmap (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))
unfolding *pfilt-nmap* **by** *simp*
have *2*: (*pfilt* (*nell1*) ((*lsum* (*NNil nell1*) *0*))) = *nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*))
unfolding *pfilt-nmap* **by** *simp*
have *3*: *nlength* (*nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))) =
nlength (*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))
by (*simp add*: *lsum-nlength*)
have *4*: ⋀*j*. *j* ≤ *nlength* (*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*))) ⟶
nnth ((*nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*))))) *j* =
nnth ((*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))) *j*
by (*metis assms*(*1*) *assms*(*4*) *lsum-NNil ndropn-nfuse ndropn-nnth nellist.simps*(*14*) *plus-nat.add-0*)
have *5*: *nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*))) =
(*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))
using *3 4*
nellist-eq-nnth-eq[*of nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))
(*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))]
by *presburger*
show *?thesis* **using** *5 1 2*
by *force*
qed

lemma *pfilt-nfusecat-lsum-a*:

assumes *nlastnfirst* (*NCons nell nells*)
all-gr-zero nells
all-nfinite nells
nlength nell > *0*
nfinite nell
shows (*pfilt* (*nfuse nell* (*nfusecat nells*)) (*lsum nells* (*the-enat* (*nlength nell*)))) =
(*pfilt* (*nfusecat nells*) ((*lsum nells 0*)))
proof −
have *1*: (*pfilt* (*nfuse nell* (*nfusecat nells*)) (*lsum nells* (*the-enat* (*nlength nell*)))) =
nmap (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*)))
unfolding *pfilt-nmap* **by** *simp*
have *2*: (*pfilt* (*nfusecat nells*) ((*lsum nells 0*))) = *nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*))
unfolding *pfilt-nmap* **by** *simp*
have *3*: *nlength* (*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*)))) =
nlength (*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))
by (*simp add*: *lsum-nlength*)
have *4*: ⋀*j*. (*enat j*) ≤ *nlength* (*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*))) ⟹
nnth ((*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*))))) *j* =
nnth ((*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))) *j*
proof −
fix *j*
assume *a*: (*enat j*) ≤ *nlength* (*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))
show *nnth* ((*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*))))) *j* =
nnth ((*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))) *j*
proof −
have *5*: *nnth* ((*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*))))) *j* =
nnth (*nfuse nell* (*nfusecat nells*)) (*nnth* (*lsum nells* (*the-enat* (*nlength nell*))) *j*)
using *nnth-nmap*[*of j* (*lsum nells* (*the-enat* (*nlength nell*))) (*nnth* (*nfuse nell* (*nfusecat nells*)))]
a 3 **by** *auto*
have *6*: (*nnth* (*lsum nells* (*the-enat* (*nlength nell*))) *j*) =
the-enat (*nlength nell*) + (∑ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))
using *lsum-nnth*[*of j nells* (*the-enat* (*nlength nell*))]
by (*metis a assms*(*3*) *lsum-nlength nlength-nmap*)
have *7*: *nnth* ((*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))) *j* =
nnth (*nfusecat nells*) (*nnth* (*lsum nells* (*0*::*nat*)) *j*)
using *nnth-nmap*[*of j* (*lsum nells* (*0*::*nat*)) (*nnth* (*nfusecat nells*))] **using** *a* **by** *auto*
have *8*: (*nnth* (*lsum nells* (*0*::*nat*)) *j*) = (∑ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))
by (*metis* (*no-types*, *lifting*) *6 a add-0 add-diff-cancel-left*′ *assms*(*3*) *lsum-nlength*
lsum-nnth nlength-nmap)
have *9*: *nlast nell* = *nfirst* (*nfusecat nells*)
by (*metis assms*(*1*) *nfirst-nfusecat-nfirst nlastnfirst-LCons*)
have *10*: (*the-enat* (*nlength nell*) + (∑ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))) ≤
nlength (*nfuse nell* (*nfusecat nells*))
proof (*cases nfinite nells*)
case *True*
then show *?thesis*
proof −
have *101*: *j* ≤ *nlength nells*
by (*metis a lsum-nlength nlength-nmap*)
have *11*: *nlength* (*nfusecat nells*) =

```
            (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
      using nfusecat-nlength-nfinite[of nells ]
       by (metis True assms(1) assms(3) nlastnfirst-LCons)
     have 12: nlength (nfuse nell (nfusecat nells)) =
            nlength nell + (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
      using nfuse-nlength[of nell nfusecat nells ] 11 by presburger
     have 13: (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k))) =
            (the-enat (∑ k::nat = 0::nat..j. (nlength (nnth nells k))))
      by (metis 101 assms(3) sum-the-enat)
     have 14: j< the-enat (nlength nells) ⟹
            (∑ k::nat = 0::nat..j. (nlength (nnth nells k))) ≤
            (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
       by (metis bot-nat-0.extremum canonically-ordered-monoid-add-class.lessE
          enat-le-plus-same(1) sum.ub-add-nat)
     have 141: j= the-enat (nlength nells) ⟹
            (∑ k::nat = 0::nat..j. (nlength (nnth nells k))) ≤
            (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
       by blast
     have 142: (∑ k::nat = 0::nat..j. (nlength (nnth nells k))) ≤
            (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
      using 14 141 101 True nfinite-nlength-enat by fastforce
     have 143: enat (the-enat (nlength nell)) = nlength nell
        by (simp add: assms(5) enat-the-enat nfinite-nlength-enat)
     have 144: enat (the-enat (∑ k = 0..j. nlength (nnth nells k))) =
            (∑ k = 0..j. nlength (nnth nells k))
        by (metis 11 142 True assms(1) assms(3) dual-order.refl enat-ord-code(4)
           enat-the-enat leD nfinite-nlength-enat nfusecat-nlength-nfinite
            nlastnfirst-LCons sum-finite)
     have 143: enat (the-enat (nlength nell) + (∑ k = 0..j. the-enat (nlength (nnth nells k)))) =
              nlength nell + (∑ k = 0..j. nlength (nnth nells k))
        using 13
        by (metis 143 144 plus-enat-simps(1))
     have 15: (the-enat (nlength nell) + (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k)))) ≤
            nlength nell + (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
      using 142 6 8 13 by (metis 143 add-left-mono)
     show ?thesis
     using 12 15 by presburger
   qed
  next
  case False
  then show ?thesis
     proof −
      have 200: ¬nfinite (nfuse nell (nfusecat nells))
       using assms by (metis False nfuse-nfinite nfusecat-nfinite nlastnfirst-LCons)
      show ?thesis
      by (metis 200 linorder-le-cases nfinite-ntaken ntaken-all)
     qed
  qed
have 30: nnth (nfuse nell (nfusecat nells)) (the-enat (nlength nell) +
       (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k)))) =
```

```
        nnth (nfusecat nells) (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k)))
      by (metis 9 assms(5) ndropn-nfuse ndropn-nnth)
    show ?thesis
    using 30 5 6 7 8 by presburger
   qed
  qed
  show ?thesis
  by (metis 3 4 nellist-eq-nnth-eq pfilt-nmap)
qed

lemma pfilt-nfusecat-lsum:
assumes nlastnfirst (NCons nell nells)
        all-gr-zero nells
        all-nfinite nells
         nlength nell > 0
        nfinite nell
shows  (pfilt (nfusecat (NCons nell nells)) (addzero (lsum (NCons nell nells) 0))) =
       (NCons (nfirst nell) (NCons  (nlast nell)   (pfilt (nfusecat nells) (lsum nells 0))))
proof −
 have 1:  nfusecat (NCons nell nells) = nfuse nell (nfusecat nells)
   by simp
 have 2: (pfilt (nfusecat (NCons nell nells)) (addzero (lsum (NCons nell nells) 0))) =
         (pfilt (nfuse nell (nfusecat nells)) (addzero (lsum (NCons nell nells) 0)))
  using 1 by simp
 have 3: addzero (lsum (NCons nell nells) 0) =
        (NCons 0 (NCons (the-enat (nlength nell))  (lsum nells (the-enat (nlength nell)))))
   using lsum-addzero-NCons by auto
 have 4: (pfilt (nfuse nell (nfusecat nells)) (addzero (lsum (NCons nell nells) 0))) =
         (pfilt (nfuse nell (nfusecat nells)) (NCons 0 (NCons (the-enat (nlength nell))
               (lsum nells (the-enat (nlength nell))))))
   using 3 by auto
 have 5:  (pfilt (nfuse nell (nfusecat nells)) (NCons 0 (NCons (the-enat (nlength nell))
            (lsum nells (the-enat (nlength nell)))))) =
           (NCons (nnth (nfuse nell (nfusecat nells)) 0)
           (NCons (nnth (nfuse nell (nfusecat nells)) (the-enat (nlength nell)))
           (pfilt (nfuse nell (nfusecat nells)) (lsum nells (the-enat (nlength nell))))))
   by simp
 have 6: (nnth (nfuse nell (nfusecat nells)) 0) = (nnth nell 0)
   using assms
   by (metis nfirst-nfusecat-nfirst nfuse-nnth nlastnfirst-LCons zero-enat-def zero-le)
 have 7: (nnth (nfuse nell (nfusecat nells)) (the-enat (nlength nell))) =
         (nnth nell (the-enat (nlength nell)))
   using assms
   by (metis ndropn-nfirst ndropn-nfuse nfirst-nfusecat-nfirst nlastnfirst-LCons nnth-nlast)
 have 8: nidx (addzero (lsum  nells 0))
   using assms lsum-addzero-nidx by blast
 have 9: (pfilt (nfuse nell (nfusecat nells)) (lsum nells (the-enat (nlength nell)))) =
         (pfilt (nfusecat nells) (lsum nells 0))
    using assms pfilt-nfusecat-lsum-a by blast
   show ?thesis
```

using *3 6 7 9*
by (*metis 2 5 assms(5) nlast-NNil nnth-nlast ntaken-0 ntaken-nlast*)
qed

lemma *pfilt-nfusecat-lsum-1*:
assumes *nlastnfirst* (*NCons nell nells*)
all-gr-zero nells
all-nfinite nells
nlength nell > *0*
nfinite nell
shows (*pfilt* (*nfusecat* (*NCons nell nells*)) ((*lsum* (*NCons nell nells*) *0*))) =
(*NCons* (*nlast nell*) (*pfilt* (*nfusecat nells*) (*lsum nells 0*)))
using *assms*
pfilt-nfusecat-lsum[*of nell nells*]
by (*metis lsum-addzero-NCons nellist.inject*(*2*) *pfilt-code*(*2*))

lemma *pfilt-nfusecat-lsum-2*:
assumes *nlastnfirst* (*nells*)
all-gr-zero nells
all-nfinite nells
j ≤ *nlength nells*
shows (*nnth* (*pfilt* (*nfusecat* (*nells*)) ((*lsum* (*nells*) *0*))) *j*) = *nlast*(*nnth nells j*)
proof −
have *1*: (*pfilt* (*nfusecat* (*nells*)) ((*lsum* (*nells*) *0*))) =
nmap (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*))
using *pfilt-nmap*[*of* (*nfusecat* (*nells*)) (*lsum* (*nells*) *0*)] **by** *simp*
have *2*: (*nnth* (*pfilt* (*nfusecat* (*nells*)) ((*lsum* (*nells*) *0*))) *j*) =
nnth (*nfusecat nells*) (*nnth* (*lsum nells* (*0*::*nat*)) *j*)
using *1 nnth-nmap*[*of j* ((*lsum* (*nells*) *0*)) (*nnth* (*nfusecat nells*))]
by (*simp add*: *assms*(*4*) *lsum-nlength*)
have *3*: (*nnth* (*lsum nells* (*0*::*nat*)) *j*) = (∑ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))
by (*metis add-0 assms*(*3*) *assms*(*4*) *lsum-nnth*)
have *4*: (∑ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*))) =
the-enat ((∑ *k*::*nat* = *0*::*nat*..*j*. (*nlength* (*nnth nells k*))))
by (*metis assms*(*3*) *assms*(*4*) *sum-the-enat*)
have *40*: *nfinite* (*ntaken j nells*)
by *simp*
have *41*: *nlastnfirst* (*ntaken j nells*)
by (*simp add*: *assms*(*1*) *assms*(*4*) *nlastnfirst-ntaken*)
have *42*: *all-nfinite* (*ntaken j nells*)
by (*metis assms*(*3*) *nset-ntaken subset-iff*)
have *5*: *nnth* (*nfusecat nells*) (*the-enat* ((∑ *k*::*nat* = *0*::*nat*..*j*. (*nlength* (*nnth nells k*))))) =
nlast(*nnth nells j*)
using *nlastfirst-nfusecat-nlast*[*of ntaken j nells*] *nfusecat-ntake*[*of j nells*] *assms*
nlastnfirst-ntaken[*of j nells*] *ntake-eq-ntaken ntaken-nlast*[*of j nells*]
by (*metis 3 4 40 42 enat-ord-simps*(*4*) *enat-the-enat ntaken-nlast sum-finite*)
show *?thesis*
using *2 3 4 5* **by** *presburger*
qed

lemma *pfilt-nfusecat-lsum-3*:
assumes *nlastnfirst* (*nells*)
all-gr-zero nells
all-nfinite nells
j ≤ *nlength* (*addzero* (*lsum nells 0*))
shows (*j=0* ⟶ (*nnth* (*pfilt* (*nfusecat* (*nells*)) (*addzero*(*lsum* (*nells*) *0*))) *j*) = *nfirst*(*nfirst nells*))
∧
(*j>0* ⟶ (*nnth* (*pfilt* (*nfusecat* (*nells*)) (*addzero*(*lsum* (*nells*) *0*))) *j*) = *nlast*(*nnth nells* (*j−1*)))

using *assms*
proof −
have *1*: *j* ≤ *nlength* (*addzero* (*lsum nells 0*)) ∧ *j=0* ⟶
(*nnth* (*pfilt* (*nfusecat* (*nells*)) (*addzero*(*lsum* (*nells*) *0*))) *j*) = *nfirst*(*nfirst nells*)
using *assms*
by (*metis lsum-addzero-nfirst nfinite-ntaken nfirst-nfusecat-nfirst nnth-NNil nnth-nlast*
ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth)
have *2*: *j* ≤ *nlength* (*addzero* (*lsum nells 0*)) ∧ *j>0* ⟶
(*nnth* (*pfilt* (*nfusecat nells*) (*addzero*(*lsum* (*nells*) *0*))) *j*) =
(*nnth* (*nfusecat nells*) (*nnth* (*addzero*(*lsum* (*nells*) *0*)) *j*))
by (*simp add*: *pfilt-nlength pfilt-nnth*)
have *3*: *j* ≤ *nlength* (*addzero* (*lsum nells 0*)) ∧ *j>0* ⟶
nnth (*addzero*(*lsum* (*nells*) *0*)) *j* = *nnth* (*lsum nells 0*) (*j−1*)
by (*metis Suc-diff-1 addzero-def enat-0-iff*(*1*) *le-zero-eq nat-less-le nnth-Suc-NCons*)
have *20*: *j* ≤ *nlength* (*addzero* (*lsum nells 0*)) ∧ *j>0* ⟶ *enat* ((*j*::*nat*) − (*1*::*nat*)) ≤ *nlength nells*
by (*metis Suc-diff-1 Suc-ile-eq addzero-def dual-order.strict-trans1 enat-0-iff*(*1*) *iless-Suc-eq*
lsum-nlength nlength-NCons not-gr-zero)
have *4*: *j* ≤ *nlength* (*addzero* (*lsum nells 0*)) ∧ *j>0* ⟶
(*nnth* (*nfusecat nells*) (*nnth* (*lsum nells 0*) (*j−1*))) = *nlast*(*nnth nells* (*j* −*1*))
using *assms 20 pfilt-nfusecat-lsum-2*[*of nells j−1*]
by (*metis lsum-nlength pfilt-expand*)
show *?thesis*
using *1 2 3 4*
by (*simp add*: *assms*(*4*))
qed

lemma *pfilt-nfusecat-lsum-4*:
assumes *nlastnfirst nells*
all-gr-zero nells
all-nfinite nells
nfinite nells
shows (*nnth* (*addzero* (*lsum nells 0*)) (*the-enat* (*nlength* (*addzero* (*lsum nells 0*))))) ≤ *nlength* (*nfusecat nells*)
proof −
have *2*: *nlength* (*nfusecat nells*) = (∑ *i* = *0..the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
using *assms nfusecat-nlength-nfinite* **by** *blast*
have *3*: *nlast* (*lsum nells 0*) =
(∑ *k* = *0..the-enat* (*nlength nells*). *the-enat* (*nlength* (*nnth nells k*)))

```
   by (simp add: assms(3) assms(4) lsum-nlast)
 have 4: (∑ k = 0..the-enat (nlength nells). the-enat (nlength (nnth nells k))) =
        the-enat(∑ k = 0..the-enat (nlength nells). (nlength (nnth nells k)))
    by (simp add: assms(3) assms(4) enat-the-enat nfinite-nlength-enat sum-the-enat)
 have 5: enat (∑ k = 0..the-enat (nlength nells). the-enat (nlength (nnth nells k))) ≤
        (∑ i = 0..the-enat (nlength nells). nlength (nnth nells i))
   using assms by (metis 4 dual-order.refl enat-ord-code(3) enat-the-enat)
 have 6: nlast (addzero (lsum nells 0)) ≤ nlength (nfusecat nells)
    unfolding addzero-def using  3 2 5
    by simp
 have 7: nlast (addzero (lsum nells 0)) =
        (nnth (addzero (lsum nells 0)) (the-enat (nlength (addzero (lsum nells 0)))))
  by (simp add: addzero-def assms(4) nfinite-lsum-conv-a nnth-nlast)
 show ?thesis using assms 6 7 by auto
qed

lemma pfilt-nfusecat-lsum-5:
 assumes nlastnfirst nells
        all-gr-zero nells
        all-nfinite nells
        (enat i) ≤ nlength (addzero (lsum nells 0))
        nfinite nells
 shows    (nnth (addzero (lsum nells 0)) i) ≤ nlength (nfusecat nells)
using assms
proof −
 have 2: nlength (nfusecat nells) = (∑ i = 0..the-enat (nlength nells). nlength (nnth nells i))
   using  assms nfusecat-nlength-nfinite by blast
 have 3:  nlength nells > 0 ⟹ nlength (lsum nells 0) >0
    by (simp add: lsum-nlength)
 have 4: nlength nells > 0 ⟹ (nnth (addzero (lsum nells 0)) i) = nnth (NCons 0 (lsum nells 0)) i
  using assms unfolding addzero-def using 3 by simp
 have 5: nlength nells > 0 ∧ i=0 ⟹ (nnth (addzero (lsum nells 0)) i) = 0
    using 4 by auto
 have 6: nlength nells > 0 ∧ i>0 ⟹ (nnth (addzero (lsum nells 0)) i) = nnth (lsum nells 0) (i −1)
    by (metis 4 Suc-diff-1 nnth-Suc-NCons)
 have 7: nlength nells > 0 ∧ i>0 ⟹ nnth (lsum nells 0) (i −1) =
          (∑ k = 0..(i−1). the-enat (nlength (nnth nells k)))
     using assms lsum-nnth[of i−1 nells 0]
     by (metis Suc-diff-1 Suc-ile-eq add-0 addzero-def i0-less iless-Suc-eq lsum-nlength
        nlength-NCons)
 have 8: nlength nells > 0 ∧ i>0 ⟹
        (∑ k = 0..(i−1). the-enat (nlength (nnth nells k))) =
        the-enat(∑ k = 0..(i−1). (nlength (nnth nells k)))
     using assms sum-the-enat[of nells i−1  ]
     by (metis Suc-diff-1 Suc-ile-eq addzero-def i0-less iless-Suc-eq lsum-nlength nlength-NCons)
 have 9: nlength nells > 0 ∧ i>0 ⟹ nnth (lsum nells 0) (i −1) ≤  nlength (nfusecat nells)
     using assms nfusecat-nlength-b[of nells i 0] pfilt-nfusecat-lsum-4[of nells]
      3 4 6 lsum-nlength[of nells] unfolding addzero-def by simp
      (metis iless-Suc-eq order.order-iff-strict the-enat.simps zero-enat-def zero-le)
 have 10:  nlength nells = 0 ∧ i = 0 ⟹  nnth (NCons 0 (lsum nells 0)) i ≤ nlength (nfusecat nells)
```

```
    using assms zero-enat-def by auto
 have 11:  nlength nells = 0 ∧ i = 1 ⟹  nnth (NCons 0 (lsum nells 0)) i ≤ nlength (nfusecat nells)
  using assms
  by (metis addzero-def lsum-nlength nlength-NCons not-one-le-zero one-eSuc one-enat-def
      pfilt-nfusecat-lsum-4 the-enat.simps)
 have 12: nfinite (nfirst nells)
    by (metis assms(3) assms(5) nconcat-expand nfinite-nappend nfinite-nconcat)
 have 13: nlength (nfirst nells) > 0
  using assms
  by (metis in-nset-conv-nnth nlast-NNil ntaken-0 ntaken-nlast zero-enat-def zero-le)
 have 14: nlength nells = 0  ⟹ i ≤ 1
    using assms  lsum-addzero-nlength[of nells]
    by (metis 12 13 enat-ord-simps(1) one-enat-def)
 have 15: nnth (NCons 0 (lsum nells 0)) i ≤ nlength (nfusecat nells)
   by (metis 10 11 14 4 5 6 9 One-nat-def Suc-leI dual-order.antisym not-gr-zero
       zero-enat-def zero-le)
 show ?thesis
 by (metis 12 13 15 assms(4) gr-zeroI lsum-addzero-nnth)
qed

lemma pfilt-nfusecat-lsum-5-alt:
 assumes nlastnfirst (nells)
         all-gr-zero nells
         all-nfinite nells
          (enat i) ≤ nlength (addzero (lsum nells 0))
         ¬nfinite nells
 shows    (nnth (addzero (lsum nells 0)) i) ≤ nlength (nfusecat nells)
using assms
by (metis enat-ile linorder-le-cases nfinite-conv-nlength-enat nfusecat-nfinite)


lemma lcppl-lsum-less-th-equal:
 assumes nidx  ls
         nnth ls 0 = 0
         nfinite ls
         nfinite σ
          nlast ls  = (the-enat (nlength σ))
          (∀  i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g)
          nlength σ > 0
          i < nlength (addzero (lsum (lcppl f g σ ls) 0))
 shows    (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) ≤
           nlength( (nfusecat (lcppl f g σ ls)))
proof −
 have 1: nlastnfirst (lcppl f g σ ls)
    using assms
    by (metis enat.distinct(2) enat-the-enat i0-less lcppl-nfusecat-nlastnfirst
        nfinite-conv-nlength-enat nnth-nlast the-enat-0)
 have 2: all-gr-zero  (lcppl f g σ ls)
  using assms all-gr-zero-nnth-a lcppl-nlength-all-gr-zero by blast
 have 3: all-nfinite (lcppl f g σ ls)
```

using *assms all-nfinite-nnth-a lcppl-nlength-all-nfinite* **by** *blast*
have *4*: *enat* (*Suc* (*i*::*nat*)) ≤ *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) (*0*::*nat*)))
using *assms* **using** *Suc-ile-eq* **by** *blast*
have *5*: *nfinite* (*lcppl f g σ ls*)
using *assms* **using** *lcppl-nfinite* **by** *blast*
show *?thesis* **using**
pfilt-nfusecat-lsum-5[*of* (*lcppl f g σ ls*) *Suc i*]
using *1 2 3 4 5* **by** *blast*
qed

lemma *lcppl-lsum-less-th-equal-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬*nfinite ls*
¬*nfinite σ*
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
i < *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
shows (*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) ≤
nlength((*nfusecat* (*lcppl f g σ ls*)))
proof −
have *1*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *2*: *all-gr-zero* (*lcppl f g σ ls*)
using *assms all-gr-zero-nnth-a lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *3*: *all-nfinite* (*lcppl f g σ ls*)
using *assms all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt* **by** *blast*
have *4*: *enat* (*Suc* (*i*::*nat*)) ≤ *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) (*0*::*nat*)))
using *assms* **using** *Suc-ile-eq* **by** *blast*
show *?thesis*
using *1 2 3 4 pfilt-nfusecat-lsum-5-alt* **using** *assms*(*1*) *assms*(*3*) *lcppl-nfinite* **by** *blast*
qed

lemma *lcppl-lsum-nlength*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = (*the-enat* (*nlength σ*))
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ >*0*
shows *nlength* (*addzero* (*lsum* ((*lcppl f g σ ls*)) *0*)) = *nlength ls*
proof −
have *1*: *nlength σ* >*0* ⟶ *nlength* (*lcppl f g σ ls*) = *nlength ls* −*1*
using *assms*
by (*metis epred-0 epred-conv-minus i0-less lcppl-nlength lcppl-nlength-zero*)
have *2*: *nlength σ*>*0* ⟶ *nlength ls* >*0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)

have *3*: *nlength σ > 0 ⟶ nlength (nfirst (lcppl f g σ ls)) > 0*
using *assms*
by (*metis lcppl-nlength-all-gr-zero ndropn-0 ndropn-nfirst nfinite-nlength-enat the-enat.simps zero-enat-def zero-le*)
have *30*: *nlastnfirst (lcppl f g σ ls)*
using *2 assms lcppl-nfusecat-nlastnfirst* **by** *blast*
have *31*: *nfinite (nfirst (lcppl f g σ ls))*
using *assms*
by (*metis all-nfinite-nnth-a lcppl-nfinite lcppl-nlength-all-nfinite nconcat-expand nfinite-nappend nfinite-nconcat*)
have *4*: *nlength σ >0 ⟶ nlength (lcppl f g σ ls) = 0 ⟶ nlength (addzero (lsum (lcppl f g σ ls) 0)) = nlength ls*
using *1 2 3 assms lsum-addzero-nlength*[*of (lcppl f g σ ls)*] *30 31*
by (*metis co.enat.exhaust-sel i0-less lcppl-nlength one-eSuc*)
have *5*: *nlength σ >0 ⟶ nlength (lcppl f g σ ls) > 0 ⟶ nlength (addzero (lsum ((lcppl f g σ ls)) 0)) = nlength ls*
using *assms*
by (*metis 2 4 addzero-def co.enat.exhaust-sel lcppl-nlength lcppl-nlength-zero lsum-nlength nlength-NCons*)
show *?thesis* **using** *4 5*
using *assms*(*7*) *gr-zeroI* **by** *blast*
qed

lemma *lcppl-lsum-nlength-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
shows *nlength (addzero (lsum ((lcppl f g σ ls)) 0)) = nlength ls*
proof −
have *1*: *nlength (lcppl f g σ ls) = nlength ls −1*
using *assms*
by (*simp add*: *epred-conv-minus lcppl-nlength-alt*)
have *2*: *nlength ls >0*
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *3*: *nlength (nfirst (lcppl f g σ ls)) > 0*
using *assms lcppl-nlength-all-gr-zero-alt*[*of ls σ f g*]
by (*metis ndropn-0 ndropn-nfirst zero-enat-def zero-le*)
have *4*: *nlength (lcppl f g σ ls) = 0 ⟶ nlength (addzero (lsum ((lcppl f g σ ls)) 0)) = nlength ls*
using *1 2 3 assms*
by (*metis lcppl-nfinite nlength-eq-enat-nfiniteD zero-enat-def*)
have *5*: *nlength (lcppl f g σ ls) > 0 ⟶ nlength (addzero (lsum ((lcppl f g σ ls)) 0)) = nlength ls*
using *assms*
by (*metis 4 addzero-def co.enat.exhaust-sel lcppl-nlength-alt lcppl-nlength-zero lsum-nlength nlength-NCons*)
show *?thesis* **using** *4 5*

using *assms gr-zeroI* **by** *blast*
qed

lemma *lcppl-lsum-nnth*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
nlength σ > 0
j ≤ nlength (addzero (lsum (lcppl f g σ ls) 0))
shows *(j=0 ⟶*
(nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
nfirst(nfirst (lcppl f g σ ls)))
∧
(j>0 ⟶ (nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
nlast(nnth (lcppl f g σ ls) (j−1)))

proof −
have *0*: *nlength σ >0 ⟶ nlength ls >0*
using *assms*
by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *1*: *nlength σ >0 ⟶ nlastnfirst (lcppl f g σ ls)*
using *0 assms*
by (*simp add*: *enat-the-enat lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat*)
have *2*: *nlength σ>0 ⟶*
(∀ j ≤ nlength (lcppl f g σ ls). nlength(nnth (lcppl f g σ ls) j) > 0)
using *assms*
by (*metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have *3*: *nlength σ>0 ⟶*
nlength (addzero (lsum (lcppl f g σ ls) 0)) =
nlength (lcppl f g σ ls) +1
using *assms*
by (*metis 0 co.enat.exhaust-sel i0-less lcppl-lsum-nlength lcppl-nlength plus-1-eSuc(2)*)
have *4*: *nlength σ > 0 ⟶*
(nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) 0) =
nfirst(nfirst (lcppl f g σ ls))
by (*metis lsum-addzero-nfirst nfirst-nfusecat-nfirst nlast-NNil ntaken-0 ntaken-nlast pfilt-nnth zero-enat-def zero-le*)
have *5*: *nlength σ > 0 ⟶*
j ≤ nlength (addzero (lsum (lcppl f g σ ls) 0)) ∧ j>0 ⟶
(nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) (j))
= nlast(nnth (lcppl f g σ ls) (j−1))
using *assms pfilt-nfusecat-lsum-3*[*of* (*lcppl f g σ ls*) *j*]
by (*metis 1 2 all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite*)
from *4 5* **show** *?thesis* **using** *assms*
by *blast*
qed

lemma *lcppl-lsum-nnth-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
j ≤ nlength (addzero (lsum (lcppl f g σ ls) 0))
shows *(j=0 ⟶*
(nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
nfirst(nfirst (lcppl f g σ ls)))
∧
(j>0 ⟶ (nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
nlast(nnth (lcppl f g σ ls) (j−1)))
proof −
have *1*: *nlastnfirst (lcppl f g σ ls)*
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *2*: *(∀ j ≤ nlength (lcppl f g σ ls). nlength(nnth (lcppl f g σ ls) j) > 0)*
using *assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *3*: *nlength (addzero (lsum (lcppl f g σ ls) 0)) = nlength (lcppl f g σ ls) +1*
using *assms*
by (*simp add*: *lcppl-lsum-nlength-alt lcppl-nlength-alt nfinite-conv-nlength-enat*)
have *4*: *(nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) 0) =*
nfirst(nfirst (lcppl f g σ ls))
by (*metis lsum-addzero-nfirst nfirst-nfusecat-nfirst nlast-NNil ntaken-0 ntaken-nlast pfilt-nnth zero-enat-def zero-le*)
have *5*: *j ≤ nlength (addzero (lsum (lcppl f g σ ls) 0)) ∧ j>0 ⟶*
(nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) (j))
= nlast(nnth (lcppl f g σ ls) (j−1))
using *assms pfilt-nfusecat-lsum-3*[*of (lcppl f g σ ls) j*]
using *1 2 all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt* **by** *blast*
from *4 5* **show** *?thesis* **using** *assms*
by *blast*
qed

lemma *lcppl-lsum-nnth-a*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
nlength σ>0
j ≤ nlength ls
shows *(nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =*
(nnth ls j)
proof −
have *1*: *nlength σ>0 ⟶ nlength ls>0*
using *assms*
by (*metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0*

```
    zero-less-iff-neq-zero)
 have 2: nlength σ>0 ⟶ nlength (addzero (lsum (lcppl f g σ ls) 0)) = nlength ls
   by (simp add: assms lcppl-lsum-nlength)
 have 3: nlength σ>0 ⟶
        j ≤ nlength ls ⟶
          ( j=0 ⟶
        (nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
         nfirst(nfirst (lcppl f g σ ls)) )
        ∧
        (j>0 ⟶ (nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
         nlast(nnth (lcppl f g σ ls) (j−1)))
    by (metis 2 assms lcppl-lsum-nnth)
 have 4: nlength σ>0 ⟶
        j ≤ nlength ls ∧ j=0 ⟶ nfirst(nfirst (lcppl f g σ ls)) = (nnth ls j)
    using assms lcppl-nfirst[of ls σ f g]
    by (metis 1  nlast-NNil ntaken-0 ntaken-nlast)
 have 5: nlength σ>0 ⟶
        j ≤ nlength ls ∧ j>0 ⟶ j−1 ≤ nlength (lcppl f g σ ls)
     using assms
     by (metis 1 Suc-diff-1 Suc-ile-eq co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength)
 have 6: nlength σ>0 ⟶
        j ≤ nlength ls ∧ j>0 ⟶ nlast(nnth (lcppl f g σ ls) (j−1)) = (nnth ls j)
   using lcppl-nlast-nnth assms
   by (metis 5 Suc-diff-1  enat.distinct(2) enat-the-enat nfinite-conv-nlength-enat)
 show ?thesis
 using 3 4 6 using assms
 by (metis not-gr-zero)
qed

lemma lcppl-lsum-nnth-a-alt:
 assumes nidx  ls
        nnth ls 0 = 0
        ¬nfinite ls
        ¬nfinite σ
        (∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g)
        j ≤ nlength ls
 shows   (nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
        (nnth ls j)
proof −
 have 2:  nlength (addzero (lsum (lcppl f g σ ls) 0)) = nlength ls
   by (simp add: assms lcppl-lsum-nlength-alt)
 have 3: j ≤ nlength ls ⟶
          ( j=0 ⟶
        (nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
         nfirst(nfirst (lcppl f g σ ls)) )
        ∧
        (j>0 ⟶ (nnth (pfilt (nfusecat ((lcppl f g σ ls))) (addzero(lsum ((lcppl f g σ ls)) 0))) j) =
         nlast(nnth (lcppl f g σ ls) (j−1)))
    by (metis 2 assms lcppl-lsum-nnth-alt)
 have 4: j ≤ nlength ls ∧ j=0 ⟶ nfirst(nfirst (lcppl f g σ ls)) = (nnth ls j)
```

using *assms lcppl-nfirst-alt*[*of ls σ f g*] **by** (*metis ndropn-0 ndropn-nfirst*)
have *5*: *j* ≤ *nlength ls* ∧ *j*>*0* ⟶ *j*−*1* ≤ *nlength* (*lcppl f g σ ls*)
using *assms*
by (*meson enat-ile lcppl-nfinite linorder-le-cases nfinite-conv-nlength-enat*)
have *6*: *j* ≤ *nlength ls* ∧ *j*>*0* ⟶ *nlast*(*nnth* (*lcppl f g σ ls*) (*j*−*1*)) = (*nnth ls j*)
using *lcppl-nlast-nnth-alt assms*
by (*metis 5 Suc-pred'*)
show *?thesis*
using *3 4 6* **using** *assms*
by (*metis not-gr-zero*)
qed

lemma *lcppl-pfilt-nfusecat-lsum*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ>*0*
shows (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *ls*
using *assms*
proof −
have *0*: *nlength σ*>*0* ⟶ *nlength ls* > *0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *1*: *nlength σ*>*0* ⟶
nlength (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *nlength ls*
using *assms*
by (*simp add*: *lcppl-lsum-nlength pfilt-nlength*)
have *2*: *nlength σ*>*0* ⟶
(∀ *j*. *j*≤*nlength ls* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
(*nnth ls j*))
by (*simp add*: *assms lcppl-lsum-nnth-a*)
from *1 2* **show** *?thesis* **using** *assms nellist-eq-nnth-eq*
by *metis*
qed

lemma *lcppl-pfilt-nfusecat-lsum-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite ls*
¬*nfinite σ*
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *ls*
using *assms*
proof −
have *1*: *nlength* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *nlength ls*
using *assms* **by** (*simp add*: *lcppl-lsum-nlength-alt pfilt-nlength*)

have *2*: (∀ *j*. *j*≤*nlength ls* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
(*nnth ls j*))
by (*simp add*: *assms lcppl-lsum-nnth-a-alt*)
from *1 2* **show** *?thesis* **using** *assms nellist-eq-nnth-eq*
by *metis*
qed

lemma *lcppl-nfusecat-pfilt-powerinterval*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat (*nlength σ*)
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ >*0*
shows *powerinterval g* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
proof −
have *0*: *nlength σ* > *0* ⟶ *nlength ls* >*0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *01*: (∀ *i*<*nlength ls*.
g (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *assms cppl-fprojection* **by** *auto*
have *02*: *nlength σ* > *0* ⟶ (∀ *i*<*nlength ls*. *g* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)))
using *0 assms lcppl-nfusecat-pfilt-fpower-help*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-conv-nlength-enat*)
have *03* : *powerinterval g* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) =
(∀ *i*<*nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)).
g (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
by (*simp add*: *powerinterval-def*)
have *04*: *nlength σ* >*0* ⟶ *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) = *nlength ls*
using *assms lcppl-lsum-nlength* **by** *blast*
have *05*: *nlength σ* >*0* ⟶
(∀ *i*<*nlength ls*.
(*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
(*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))

proof −
have *06*: *nlength σ* >*0* ⟶
(∀ *i*<*nlength ls*. *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
nlength (*nnth* (*lcppl f g σ ls*) *i*))
using *pfilt-nlength* **by** *blast*
have *07*: *nlength σ* >*0* ⟶

```
        (∀ i<nlength ls. nlength (nnth (lcppl f g σ ls) i) =
          nlength (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))))

    using assms by (simp add: lcppl-nnth)
have 08: nlength σ >0 ⟶
        (∀ i<nlength ls.
          nlength (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))) =
          nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))

     using nlength-nmap by blast
have 09: nlength σ >0 ⟶
        (∀ i<nlength ls.
         nlength (nsubn  (pfilt σ (nfusecat (lcppl f g σ ls)))
                         (nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
                         (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
                 ) =
         (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) −
         (nnth (addzero (lsum (lcppl f g σ ls) 0)) i) )
         using assms
       by (metis 04 enat-minus-mono1 idiff-enat-enat lcppl-lsum-less-th-equal min.orderE
    nsubn-nlength pfilt-nlength)
have 10: nlength σ >0 ⟶
        (∀ i<nlength ls. (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) =
          (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) (Suc i)))

    using 04 addzero-def by auto
have 11: nlength σ >0 ⟶
        (∀ i<nlength ls. (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) (Suc i)) =
          (∑ k::nat= 0..(i). nlength(nnth (lcppl f g σ ls) k)) )
          using 04
   proof simp-all
     assume nlength σ ≠ (0::enat) ⟶ nlength (addzero (lsum (lcppl f g σ ls) (0::nat))) = nlength ls
     show    nlength σ ≠ (0::enat) ⟶
          (∀ i::nat. enat i < nlength ls ⟶ enat (nnth (lsum (lcppl f g σ ls) (0::nat)) i) =
          (∑ k::nat = 0::nat..i. nlength (nnth (lcppl f g σ ls) k)))
      proof
       assume  nlength σ ≠ (0::enat)
       show  (∀ i::nat. enat i < nlength ls ⟶
            enat (nnth (lsum (lcppl f g σ ls) (0::nat)) i) =
            (∑ k::nat = 0::nat..i. nlength (nnth (lcppl f g σ ls) k)))
       proof
        fix i
        show i < nlength ls ⟶
             enat (nnth (lsum (lcppl f g σ ls) (0::nat)) i) =
            (∑ k::nat = 0::nat..i. nlength (nnth (lcppl f g σ ls) k))
            proof
              assume a: i < nlength ls
              show enat (nnth (lsum (lcppl f g σ ls) 0) i) =
                   (∑ k = 0..i. nlength (nnth (lcppl f g σ ls) k))
              proof −
```

have *110*: *all-nfinite* (*lcppl f g σ ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite* **by** *blast*
have *111*: *nnth* (*lsum* (*lcppl f g σ ls*) *0*) *i* =
($\sum$ *k* = *0..i*. *the-enat* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))
using *lsum-nnth*[*of i* (*lcppl f g σ ls*) *0*] *a assms*
by (*metis 0 110 add-cancel-right-left co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero*)
have *112*: ($\sum$ *k* = *0..i*. *the-enat* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*))) =
the-enat($\sum$ *k* = *0..i*. (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))
using *sum-the-enat*[*of* (*lcppl f g σ ls*) *i*] *assms*
by (*metis 0 110 a co.enat.exhaust-sel gr-implies-not-zero iless-Suc-eq lcppl-nlength*)
have *113*: ($\sum$ *k* = *0..i*. (*nlength* (*nnth* (*lcppl f g σ ls*) *k*))) < ∞
using *sum-finite*[*of* (*lcppl f g σ ls*) *i*] *0 110 assms a*
by (*metis co.enat.exhaust-sel gr-implies-not-zero iless-Suc-eq lcppl-nlength*)
have *114*: *enat* (*the-enat*($\sum$ *k* = *0..i*. (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))) =
($\sum$ *k* = *0..i*. (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))
using *113 enat-ord-simps*(*4*) *enat-the-enat* **by** *blast*
show *?thesis* **using** *111 112 114* **by** *presburger*
qed
qed
qed
qed
qed
have *12*: *nlength σ* >*0* ⟶
(∀ *i*<*nlength ls*. (*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
(*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *i*))
using *04 addzero-def* **by** *auto*
have *121*: *nlength σ* >*0* ⟶ (*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *0*) = *0*
by *simp*
have *13*: *nlength σ* >*0* ⟶
(∀ *i*<*nlength ls*. (*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
(*if i* = *0 then 0*
else ($\sum$ *k*::*nat*= *0..*(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))

using *11*
by (*metis 121 Suc-diff-1 Suc-ile-eq not-gr-zero order-less-imp-le zero-enat-def*)
have *14*: *nlength σ* >*0* ⟶
(∀ *i*<*nlength ls*.
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) −
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
($\sum$ *k*::*nat*= *0..*(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(*if i* = *0 then 0 else*
($\sum$ *k*::*nat*= *0..*(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))
using *10 11 12 13* **by** *simp* (*metis One-nat-def idiff-enat-enat not-gr-zero*)
have *15*: *nlength σ* >*0* ⟶
(∀ *i*<*nlength ls*.
($\sum$ *k*::*nat*= *0..*(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(*if i* = *0 then 0 else*
($\sum$ *k*::*nat*= *0..*(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))) =

(*if i = 0 then nlength(nnth (lcppl f g σ ls) 0) else*
(∑ k::nat= 0..(i). nlength(nnth (lcppl f g σ ls) k)) −
(∑ k::nat= 0..(i−1). nlength(nnth (lcppl f g σ ls) k))))
by (*simp add*: *Nitpick.case-nat-unfold*)
have *16*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(*if i = 0 then nlength* (*nnth* (*lcppl f g σ ls*) *0*) *else*
(∑ k::nat= 0..(i). nlength(nnth (lcppl f g σ ls) k)) −
(∑ k::nat= 0..(i−1). nlength(nnth (lcppl f g σ ls) k))) =
(*if i = 0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
nlength(*nnth* (*lcppl f g σ ls*) *i*)))
using *13 sum.cl-ivl-Suc*[*of* λ*k. nlength*(*nnth* (*lcppl f g σ ls*) *k*) *0*]
by (*metis Suc-diff-1 enat.distinct*(*2*) *enat-add-sub-same less-nat-zero-code not-gr-zero*)
have *17*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(*if i = 0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
nlength(*nnth* (*lcppl f g σ ls*) *i*)) =
nlength(*nnth* (*lcppl f g σ ls*) *i*))
by (*simp add*: *Nitpick.case-nat-unfold*)
have *18*: *nlength σ >0 ⟶*
(∀ *i<nlength ls. nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
nlength (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))

by (*simp add*: *09 14 15 16 pfilt-nlength*)
have *19*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) ≤
nlength (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))))
by (*simp add*: *04 assms lcppl-lsum-less-th-equal pfilt-nlength*)
have *22*: *nlength σ >0 ⟶ nlastnfirst* (*lcppl f g σ ls*)
using *0 assms*
by (*simp add*: *enat-the-enat lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat*)
have *23*: *nlength σ >0 ⟶* (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have *190*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) ≤
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
using *0 04 06 09 18 23 lsum-nlength*[*of* (*lcppl f g σ ls*) *0*] *lcppl-nlength*[*of ls f g σ*] *assms*
by *simp*
(*metis* (*no-types*, *lifting*) *co.enat.exhaust-sel diff-is-0-eq′ iless-Suc-eq le-cases zero-enat-def*)
have *20*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*

```
          (∀ j≤ nlength(nnth (lcppl f g σ ls) i).
            (nnth (nsubn  (pfilt σ (nfusecat (lcppl f g σ ls)))
                          (nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
                          (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
                        ) j) =
            (nnth (pfilt σ (nfusecat (lcppl f g σ ls)))
                ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) +j) ) ) )
        using nsubn-nnth[of (pfilt σ (nfusecat (lcppl f g σ ls))) ] by  simp
  (metis 06 09 18 assms(7) enat-ord-simps(1) min.orderE)
have 21: nlength σ >0 ⟶
          (∀ i<nlength ls.
            (∀ j≤ nlength(nnth (lcppl f g σ ls) i).
             (nnth (pfilt σ (nfusecat (lcppl f g σ ls)))
                 ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) +j) ) =
             (nnth σ (nnth (nfusecat (lcppl f g σ ls))
                           ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) +j) )) ) )
   using pfilt-nlength[of σ (nfusecat (lcppl f g σ ls)) ]
     pfilt-nnth[of - σ (nfusecat (lcppl f g σ ls)) ]
     by (metis nnth-0 pfilt-code(2) pfilt-pfilt)
have 24: nlength σ >0 ⟶
        (∀ i≤nlength (lcppl f g σ ls).
           (∀ j≤ nlength(nnth (lcppl f g σ ls) i).
            ( (nnth (nfusecat (lcppl f g σ ls)) ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) +j) )) =
            ( (nnth (nnth (lcppl f g σ ls) i) j))  ))

 using assms lsum-nfusecat-nnth-lsum-nnth[of (lcppl f g σ ls)] lcppl-nlength[of ls f g σ]
   0 06 09 18 22 23
   by (metis all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite)
have 241: nlength σ >0 ⟶ nlength (lcppl f g σ ls) = nlength ls −1
  using 0 assms
  by (simp add: epred-conv-minus lcppl-nlength)
have 25:  nlength σ >0 ⟶
          (∀ i<nlength ls.
            (∀ j≤ nlength(nnth (lcppl f g σ ls) i).
             (nnth σ (nnth (nfusecat (lcppl f g σ ls))
                           ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) +j) )) =
             (nnth (pfilt σ (nnth (lcppl f g σ ls) i)) j) ))
   using 0 04 24 lsum-nlength[of (lcppl f g σ ls) 0]  pfilt-nlength[of σ ]  pfilt-nnth
   by (metis addzero-def i0-less iless-Suc-eq nlength-NCons)
have 26: nlength σ >0 ⟶
          (∀ i<nlength ls.
            (∀ j≤ nlength(nnth (lcppl f g σ ls) i).
             (nnth (pfilt σ (nnth (lcppl f g σ ls) i)) j) =
             (nnth (nsubn  (pfilt σ (nfusecat (lcppl f g σ ls)))
                           (nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
                           (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
                         ) j) ) )

  by (simp add: 20 21 25)
from 18 26 show ?thesis using nellist-eq-nnth-eq
```

```
  by (metis 06)
  qed
 show ?thesis
 using 02 03 04 05 assms(7) by force
qed

lemma lcppl-nfusecat-pfilt-powerinterval-alt:
 assumes nidx ls
         nnth ls 0 = 0
         ¬nfinite ls
         ¬nfinite σ
          (∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g)
 shows      powerinterval g (pfilt σ (nfusecat (lcppl f g σ ls))) (addzero (lsum (lcppl f g σ ls) 0))
proof −
 have 01: (∀ i<nlength ls.
           g (pfilt (nsubn σ (nnth ls i) (nnth ls (Suc i))) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))))
   using assms cppl-fprojection by auto
 have 02:  (∀ i<nlength ls. g (pfilt σ (nnth (lcppl f g σ ls) i)) )
   using  assms lcppl-nfusecat-pfilt-fpower-help-alt
   by (metis  enat-the-enat nfinite-conv-nlength-enat)
 have 03 : powerinterval g (pfilt σ (nfusecat (lcppl f g σ ls))) (addzero (lsum (lcppl f g σ ls) 0)) =
          ( ∀ i<nlength (addzero (lsum (lcppl f g σ ls) 0)).
            g (nsubn (pfilt σ (nfusecat (lcppl f g σ ls)))
                     (nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
                     (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
                    ))
       by (simp add: powerinterval-def)
  have 04:   nlength (addzero (lsum (lcppl f g σ ls) 0)) = nlength ls
    using assms lcppl-lsum-nlength-alt by blast
  have 05: (∀ i<nlength ls.
             (pfilt σ (nnth (lcppl f g σ ls) i))  =
             (nsubn  (pfilt σ (nfusecat (lcppl f g σ ls)))
                  (nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
                  (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
                  ) )
   proof −
    have 06:
             (∀ i<nlength ls. nlength (pfilt σ (nnth (lcppl f g σ ls) i)) =
                   nlength (nnth (lcppl f g σ ls) i))
      using pfilt-nlength by blast
    have 07:  (∀ i<nlength ls. nlength (nnth (lcppl f g σ ls) i) =
               nlength (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))))
       using assms by (simp add: lcppl-nnth)
    have 08:  (∀ i<nlength ls.
               nlength (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))  =
               nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))
        using nlength-nmap by blast
    have 09:  (∀ i<nlength ls.
              nlength (nsubn  (pfilt σ (nfusecat (lcppl f g σ ls)))
                           (nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
```

```
                    (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
                ) =
          (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) −
          (nnth (addzero (lsum (lcppl f g σ ls) 0)) i) )
      using assms
     by (metis 04 enat-minus-mono1 idiff-enat-enat lcppl-lsum-less-th-equal-alt min-def
        nsubn-nlength pfilt-nlength)
   have 10: (∀ i<nlength ls. (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) =
          (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) (Suc i)))
     using 04 addzero-def by auto
   have 11: (∀ i<nlength ls. (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) (Suc i)) =
          (∑ k::nat= 0..(i). nlength(nnth (lcppl f g σ ls) k)) )
     using 04
     proof simp-all
      assume nlength (addzero (lsum (lcppl f g σ ls) (0::nat))) = nlength ls
      show (∀ i::nat. enat i < nlength ls ⟶ enat (nnth (lsum (lcppl f g σ ls) (0::nat)) i) =
           (∑ k::nat = 0::nat..i. nlength (nnth (lcppl f g σ ls) k)))
       proof
        fix i
        show i < nlength ls ⟶
             enat (nnth (lsum (lcppl f g σ ls) (0::nat)) i) =
            (∑ k::nat = 0::nat..i. nlength (nnth (lcppl f g σ ls) k))
            proof
              assume a: i < nlength ls
              show enat (nnth (lsum (lcppl f g σ ls) 0) i) =
                   (∑ k = 0..i. nlength (nnth (lcppl f g σ ls) k))
              proof −
              have 110: all-nfinite (lcppl f g σ ls)
                 using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt by blast
             have 111: nnth (lsum (lcppl f g σ ls) 0) i = (∑ k = 0..i. the-enat (nlength (nnth (lcppl f
g σ ls) k)))
              using lsum-nnth[of i (lcppl f g σ ls) 0] a assms
              by (metis 04 110 add-cancel-right-left addzero-def iless-Suc-eq lsum-nlength
                 nlength-NCons nlength-eq-enat-nfiniteD zero-enat-def)
              have 112: (∑ k = 0..i. the-enat (nlength (nnth (lcppl f g σ ls) k))) =
                        the-enat(∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k)))
               using sum-the-enat[of (lcppl f g σ ls) i]  assms
               by (metis 110 lcppl-nfinite linorder-le-cases nfinite-ntaken ntaken-all)
              have 113: (∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k))) < ∞
                using sum-finite[of (lcppl f g σ ls) i]  110 04 assms a
                by (metis addzero-def iless-Suc-eq lsum-nlength nlength-NCons
                   nlength-eq-enat-nfiniteD zero-enat-def)
              have 114: enat (the-enat(∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k)))) =
                        (∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k)))
                 using 113 enat-ord-simps(4) enat-the-enat by blast
              show ?thesis  using 111 112 114 by presburger
             qed
            qed
         qed
     qed
```

have *12*: (∀ *i*<*nlength ls*. (*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
(*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *i*))
using *04 addzero-def* **by** *auto*
have *121*: (*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *0*) = *0*
by *simp*
have *13*: (∀ *i*<*nlength ls*. (*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
(*if i* = *0 then 0*
else (∑ *k*::*nat*= *0*..(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))
using *11*
by (*metis 121 Suc-diff-1 Suc-ile-eq not-gr-zero order-less-imp-le zero-enat-def*)
have *14*: (∀ *i*<*nlength ls*.
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) −
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
(∑ *k*::*nat*= *0*..(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(*if i* = *0 then 0 else*
(∑ *k*::*nat*= *0*..(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))

using *10 11 12 13* **by** *simp* (*metis One-nat-def idiff-enat-enat not-gr-zero*)
have *15*: (∀ *i*<*nlength ls*.
(∑ *k*::*nat*= *0*..(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(*if i* = *0 then 0 else*
(∑ *k*::*nat*= *0*..(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))) =
(*if i* = *0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
(∑ *k*::*nat*= *0*..(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(∑ *k*::*nat*= *0*..(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))

by (*simp add*: *Nitpick.case-nat-unfold*)
have *16*: (∀ *i*<*nlength ls*.
(*if i* = *0 then nlength* (*nnth* (*lcppl f g σ ls*) *0*) *else*
(∑ *k*::*nat*= *0*..(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(∑ *k*::*nat*= *0*..(*i*−*1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))) =
(*if i* = *0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
nlength(*nnth* (*lcppl f g σ ls*) *i*)))
using *13 sum.cl-ivl-Suc*[*of* λ*k*. *nlength*(*nnth* (*lcppl f g σ ls*) *k*) *0*]
by (*metis Suc-diff-1 enat.distinct*(*2*) *enat-add-sub-same less-nat-zero-code not-gr-zero*)
have *17*: (∀ *i*<*nlength ls*.
(*if i* = *0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
nlength(*nnth* (*lcppl f g σ ls*) *i*)) =
nlength(*nnth* (*lcppl f g σ ls*) *i*))
by (*simp add*: *Nitpick.case-nat-unfold*)
have *18*: (∀ *i*<*nlength ls*. *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
nlength (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
by (*simp add*: *09 14 15 16 pfilt-nlength*)
have *19*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) ≤
nlength (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))))

by (*simp add*: *04 assms lcppl-lsum-less-th-equal-alt pfilt-nlength*)
have *22*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *23*: (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)
using *assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *190*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) ≤
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
using *04 06 09 18 23 lsum-nlength*[*of* (*lcppl f g σ ls*) *0*] *lcppl-nlength-alt*[*of ls f g σ*] *assms*
by *simp*
(*metis* (*no-types*, *opaque-lifting*) *co.enat.exhaust-sel diff-is-0-eq′ iless-Suc-eq nfinite-ntaken not-le-imp-less ntaken-all order-less-imp-le zero-enat-def*)
have *20*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
) *j*) =
(*nnth* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))))
using *nsubn-nnth*[*of* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))] **by** *simp*
(*metis 06 09 18 enat-ord-simps*(*1*) *min.orderE*)
have *21*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*)) =
(*nnth σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*)))))
using *pfilt-nlength*[*of σ* (*nfusecat* (*lcppl f g σ ls*))]
pfilt-nnth[*of - σ* (*nfusecat* (*lcppl f g σ ls*))]
by (*metis enat-ord-code*(*4*) *enat-the-enat not-le-imp-less ntaken-all ntaken-nlast order-less-imp-le*)
have *24*: (∀ *i*≤*nlength* (*lcppl f g σ ls*).
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
((*nnth* (*nfusecat* (*lcppl f g σ ls*)) ((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))) =
((*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *j*))))
using *assms lsum-nfusecat-nnth-lsum-nnth*[*of* (*lcppl f g σ ls*)] *04 22 23 all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt* **by** *blast*
have *241*: *nlength* (*lcppl f g σ ls*) = *nlength ls* −*1*
using *assms* **by** (*simp add*: *epred-conv-minus lcppl-nlength-alt*)
have *25*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))) =
(*nnth* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) *j*)))
using *04 24 lsum-nlength*[*of* (*lcppl f g σ ls*) *0*] *pfilt-nlength*[*of σ*] *pfilt-nnth assms*(*3*)
by (*metis 241 enat2-cases enat-ord-code*(*3*) *epred-Infty epred-conv-minus nfinite-conv-nlength-enat*)

```
   have 26: (∀ i<nlength ls.
              (∀ j≤ nlength(nnth (lcppl f g σ ls) i).
               (nnth (pfilt σ (nnth (lcppl f g σ ls) i)) j) =
               (nnth (nsubn  (pfilt σ (nfusecat (lcppl f g σ ls)))
                             (nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
                             (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
                          ) j) ) )
    by (simp add: 20 21 25)
  from 18 26 show ?thesis using nellist-eq-nnth-eq
  by (metis 06)
 qed
 show ?thesis
 using 02 03 04 05 assms by force
qed

lemma lcppl-nfusecat-pfilt-nlength:
 assumes nidx  ls
         nnth ls 0 = 0
         nfinite ls
         nfinite σ
          nlast ls = the-enat (nlength σ)
          (∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g)
          nlength σ >0
 shows    nlast (addzero (lsum (lcppl f g σ ls) 0))  =
          the-enat(nlength (pfilt σ (nfusecat (lcppl f g σ ls))))
proof −
 have 0:  nlength ls > 0
    using assms
    by (metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0)
 have 1: nlength (pfilt σ (nfusecat (lcppl f g σ ls))) = nlength ( (nfusecat (lcppl f g σ ls)))
  using pfilt-nlength by blast
 have 10: nfinite (lcppl f g σ ls)
    using assms(1) assms(3) lcppl-nfinite by blast
 have 11: nlastnfirst (lcppl f g σ ls)
    using assms
    by (metis 0  lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat)
 have 12:  ∀ lx ∈ nset (lcppl f g σ ls). 0 < nlength lx
    using assms
    by (metis  enat.distinct(2) enat-the-enat in-nset-conv-nnth lcppl-nlength-all-gr-zero
       nfinite-conv-nlength-enat)
 have 13: ∀ lx ∈ nset (lcppl f g σ ls). nfinite lx
  by (metis (no-types, lifting) 0 assms(1) assms(3) assms(6) co.enat.exhaust-sel
      cppl-fprojection i0-less iless-Suc-eq in-nset-conv-nnth lcppl-nlength lcppl-nnth nfinite-nmap)
 have 141:  ∀ i ≤ nlength (lcppl f g σ ls). 0 < nlength (nnth (lcppl f g σ ls) i)
    using 12 assms
    by (metis  enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat)
 have 142: ∀ i ≤ nlength (lcppl f g σ ls). nfinite (nnth (lcppl f g σ ls) i)
  using 13 assms
  by (meson in-nset-conv-nnth)
 have 15: nfinite  (nfusecat (lcppl f g σ ls))
```

using *nfusecat-nfinite*[*of* (*lcppl f g σ ls*)] *10 11 12 13* **by** *blast*
have *2*: *nlength* ((*nfusecat* (*lcppl f g σ ls*))) =
(*the-enat* (∑ *k*::*nat*= *0*..(*the-enat*(*nlength* (*lcppl f g σ ls*))). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)))
using *0 assms*
by (*metis 10 11 13 15 nfinite-conv-nlength-enat nfusecat-nlength-nfinite the-enat.simps*)
have *3*:
nlast(*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) = *nlast*((*lsum* (*lcppl f g σ ls*) *0*))
using *lsum-addzero-nlast*
using *assms*(*1*) *assms*(*3*) *lcppl-nfinite* **by** *blast*
have *4*: *nlast*((*lsum* (*lcppl f g σ ls*) *0*)) =
(∑ *k*::*nat* = *0*::*nat*..*the-enat* (*nlength* (*lcppl f g σ ls*)). *the-enat* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))
using *assms lsum-nlast*[*of* (*lcppl f g σ ls*) *0*] *10 13 plus-nat.add-0* **by** *presburger*
have *5*: (∑ *k*::*nat* = *0*..*the-enat* (*nlength* (*lcppl f g σ ls*)). *the-enat* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*))) =
(*the-enat* (∑ *k*::*nat* = *0*..*the-enat*(*nlength* (*lcppl f g σ ls*)). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)))
using *assms lsum-nnth*[*of the-enat* (*nlength* (*lcppl f g σ ls*)) (*lcppl f g σ ls*) *0*]
sum-the-enat[*of* (*lcppl f g σ ls*) *the-enat*(*nlength* (*lcppl f g σ ls*))]
by (*metis 13 dual-order.refl enat-ord-code*(*3*) *enat-the-enat*)
show *?thesis* **using** *1 2 3 4 5 assms*
by (*simp add*: *lcppl-lsum-nlength nfinite-conv-nlength-enat*)
qed

3.7.3 Soundness of Projection Axioms

lemma *PJ00sem*:
σ ⊨ ¬(*f fproj inf*)
by (*auto simp add*: *fprojection-d-def infinite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *OPJ00sem*:
σ ⊨ ¬(*f oproj finite*)
by (*auto simp add*: *oprojection-d-def finite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *PJ01sem*:
σ ⊨ (*f fproj g*) ⟶ *finite*
by (*auto simp add*: *fprojection-d-def finite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *OPJ01sem*:
σ ⊨ (*f oproj g*) ⟶ *inf*
by (*auto simp add*: *oprojection-d-def infinite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *PJ02sem*:
σ ⊨ (*f fproj g*) = ((*f* ∧ *finite*) *fproj g*)
by (*auto simp add*: *fprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)
(*metis enat-ord-simps*(*2*) *nfinite-nlength-enat nsubn-nfinite*)

lemma *OPJ02sem*:
σ ⊨ (*f oproj g*) = ((*f* ∧ *finite*) *oproj g*)
by (*auto simp add*: *oprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)

(*metis nfinite-conv-nlength-enat nfinite-ntaken nsubn-def1*)

lemma *PJ03sem*:
$\sigma \models (f \ \ fproj \ \ g) = (f \ \ fproj \ \ (g \wedge finite))$
by (*auto simp add*: *fprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)

lemma *OPJ03sem*:
$\sigma \models (f\ oproj\ g) = (f\ oproj\ (g \wedge inf))$
by (*auto simp add*: *oprojection-d-def infinite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)

PJ1

lemma *PJ1sem*:
$(\sigma \models f \ \ fproj \ \ (\ g \vee h) = (f \ \ fproj \ \ g \vee f \ \ fproj \ \ h))$
by (*simp add*: *fprojection-d-def*) *blast*

lemma *OPJ1sem*:
$(\sigma \models f\ oproj\ (\ g \vee h) = (f\ oproj\ g \vee f\ oproj\ h))$
by (*simp add*: *oprojection-d-def*) *blast*

PJ2

lemma *PJ2sem*:
$(\sigma \models f \ \ fproj \ \ empty = empty)$
proof *auto*
show $(\sigma \models f \ \ fproj \ \ empty) \Longrightarrow \sigma \models empty$
unfolding *fprojection-d-def empty-defs*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-nlength-enat nnth-nlast pfilt-nlength the-enat-0*)
show $\sigma \models empty \Longrightarrow (\sigma \models f \ \ fproj \ \ empty)$
unfolding *fprojection-d-def empty-defs powerinterval-def nidx-expand*
by (*metis enat-ord-simps*(*2*) *leD nfinite-conv-nlength-enat nlast-NNil nlength-NNil nnth-NNil not-iless0 pfilt-nlength the-enat-0 zero-enat-def zero-less-Suc*)
qed

lemma *OPJ2sem*:
$(\sigma \models \neg(f\ oproj\ empty))$
unfolding *oprojection-d-def empty-defs pfilt-nlength*
using *nlength-eq-enat-nfiniteD* **by** (*simp add*: *zero-enat-def*) *blast*

PJ3

lemma *PJ3help*:
assumes *nfinite* σ
shows $nsubn\ \sigma\ 0\ (the\text{-}enat\ (nlength\ \sigma)) = \sigma$
using *assms*
by (*metis diff-zero dual-order.refl ndropn-0 nfinite-conv-nlength-enat nsubn-def1 ntaken-all the-enat.simps*)

lemma *PJ3help1*:

assumes $f\ \sigma \wedge 0 < nlength\ \sigma \wedge nfinite\ \sigma$
shows $(\exists l.\ nidx\ \ l \wedge nnth\ l\ 0 = 0 \wedge nfinite\ l \wedge nfinite\ \sigma \wedge$
$nlast\ l = (the\text{-}enat\ (nlength\ \sigma)) \wedge$
$(\forall i{<}nlength\ l.\ f\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ l \wedge$
$(\forall i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ l\ i)) \wedge nlength\ \sigma 1 = Suc\ 0))$
proof −
have *1*: $nidx\ \ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))$
using *assms* **unfolding** *nidx-expand* **by** *simp*
(*metis Suc-ile-eq assms enat-0-iff*(*1*) *enat-ord-simps*(*2*) *iless-eSuc0 nfinite-nlength-enat nnth-0*
nnth-NNil the-enat.simps)
have *2*: $nnth\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))\ ((the\text{-}enat(nlength\ (NCons\ 0\ (NNil\ (nlength\ \sigma))))))$
$=$
$(the\text{-}enat\ (nlength\ \sigma))$
by (*metis nfinite-NCons nfinite-NNil nlast-NCons nlast-NNil nlength-NCons nlength-NNil nnth-nlast*)
have *3*: $(\forall i{<}nlength\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma)))).$
$f\ (nsubn\ \sigma\ (nnth\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))\ i)$
$(nnth\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))\ (Suc\ i)))$
using *assms PJ3help*
by (*metis enat-0-iff*(*1*) *iless-eSuc0 nlength-NCons nlength-NNil nnth-0 nnth-NNil nnth-Suc-NCons*)
have *4*: $nlength\ (NCons\ (nnth\ \sigma\ (0))\ (NNil\ (nnth\ \sigma\ (the\text{-}enat(nlength\ \sigma)))))\ =$
$nlength\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))$
by *simp*
have *5*: $(\forall i{\leq}nlength\ \ (NCons\ (nnth\ \sigma\ (0))\ (NNil\ (nnth\ \sigma\ (the\text{-}enat(nlength\ \sigma))))).$
$nnth\ \ (NCons\ (nnth\ \sigma\ (0))\ (NNil\ (nnth\ \sigma\ (the\text{-}enat(nlength\ \sigma)))))\ i =$
$nnth\ \sigma\ (nnth\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))\ i))$
by (*metis iless-Suc-eq le-zero-eq ndropn-nlast nfinite-NCons nfinite-NNil nlast-NCons*
nlength-NCons nlength-NNil nnth-0 nnth-NNil nnth-zero-ndropn order-neq-le-trans
the-enat.simps the-enat-0)
have *6*: $nlength\ (NCons\ (nnth\ \sigma\ (0))\ (NNil\ (nnth\ \sigma\ (the\text{-}enat(nlength\ \sigma)))))\ =\ Suc\ 0$
using *one-eSuc one-enat-def* **by** *auto*
have *7*: $nfinite\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))$
by *simp*
have *8*: $nlast\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))\ =\ (the\text{-}enat\ (nlength\ \sigma))$
by *simp*
have *9*: $nnth\ (NCons\ 0\ (NNil\ (the\text{-}enat(nlength\ \sigma))))\ 0 = 0$
using *nnth-0* **by** *simp*
show *?thesis*
using *1 2 3 4 5 6 7 8 9 assms* **by** *blast*
qed

lemma *PJ3sem*:
$(\sigma \models f\ \ fproj\ \ skip = (f \wedge more \wedge finite))$
proof −
have *1*: $(\sigma \models f\ \ fproj\ \ skip) \Longrightarrow (\sigma \models f \wedge more \wedge finite)$
unfolding *cppl-fprojection pfilt-expand nidx-expand skip-defs more-defs finite-defs powerinterval-def*
by (*metis* (*no-types*, *lifting*) *One-nat-def PJ3help eSuc-enat i0-less ileI1 intensional-rews*(*3*)
less-numeral-extra(*1*) *less-numeral-extra*(*3*) *nnth-nlast pfilt-nlength the-enat.simps zero-enat-def*)
have *2*: $(\sigma \models f \wedge more \wedge finite) \Longrightarrow (\sigma \models f\ \ fproj\ \ skip)$

by (*simp add*: *fprojection-d-def finite-defs skip-defs more-defs powerinterval-def pfilt-nlength*)
(*metis PJ3help1 i0-less nfinite-conv-nlength-enat the-enat.simps*)
show *?thesis*
using *1 2 unl-lift2* **by** *blast*
qed

lemma *OPJ3sem*:
($\sigma \models \neg(f$ *oproj skip*))
unfolding *oprojection-d-def skip-defs pfilt-nlength*
using *nlength-eq-enat-nfiniteD* **by** *auto*

PJ4

lemma *PJ4semchaina*:
assumes ($\sigma \models f$ *fproj* (*g*;*h*))
shows ($\sigma \models (f$ *fproj g*) ; (*f fproj h*))
proof −
have *1*: ($\sigma \models f$ *fproj* (*g*;*h*))
using *assms* **by** *auto*
have *2*: ($\exists$ *ls*. *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ *nfinite ls* $\wedge$ *nfinite* σ $\wedge$
nlast ls = (*the-enat* (*nlength* σ)) $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))))
using *assms* **unfolding** *chop-defs fprojection-d-def*
by (*metis nfinite-conv-nlength-enat pfilt-nlength the-enat.simps*)
obtain *ls* **where** *3*: *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ *nfinite ls* $\wedge$ *nfinite* σ $\wedge$
nlast ls = (*the-enat* (*nlength* σ)) $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *2* **by** *auto*
have *4*: *nidx ls* $\wedge$ *nnth ls 0* = *0*
using *3* **by** *auto*
have *5*: *powerinterval f* σ *ls*
using *3* **by** *auto*
have *6*: *nlast ls* = *the-enat* (*nlength* σ)
using *3* **by** *auto*
have *7*: ($\exists$ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *3* **by** *auto*
obtain *n* **where** *8*: *n*≤*nlength* (*pfilt* σ *ls*) $\wedge$ *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))
using *7* **by** *auto*
have *9*: *n*≤*nlength* (*pfilt* σ *ls*)
using *8* **by** *auto*
have *10*: *g* (*ntaken n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *11*: *h* (*ndropn n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *12*: *nidx* (*ntaken n ls*) $\wedge$ *nnth* (*ntaken n ls*) *0* = *0*
using *4 8* **unfolding** *nidx-expand pfilt-nlength* **by** *simp*
(*metis Suc-leD bot-nat-0.extremum dual-order.trans enat-ord-simps*(*1*) *min.orderE ntaken-nnth*)
have *13*: *nidx* (*ndropn n ls*) $\wedge$ *nnth* (*ndropn n ls*) *0* = (*nnth ls n*)

using *4 9* **unfolding** *pfilt-nlength nidx-expand*
using *nidx-expand nidx-ndropn* **by** *auto*
have *131*: $((nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ \ (ndropn\ n\ ls))) = ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls)$
using *ndropn-nmap*[*of n* $(\lambda x.\ x - (nnth\ ls\ n))$ *ls*] **by** *presburger*
have *132*: $nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))\ 0 = 0$
by (*metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le*)
have *133*: $\bigwedge j.\ enat\ j \leq nlength\ ls \Longrightarrow nnth\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls)\ j = nnth\ ls\ j - nnth\ ls\ n$
using *nnth-nmap* **by** *blast*
have *134*: $nlength\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls)) = nlength\ ls - enat\ n$
by *simp*
have *135*: $\bigwedge j.\ j \leq nlength\ ls - enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))\ j = nnth\ ls\ (n{+}\ j) - nnth\ ls\ n$
by (*metis 131 ndropn-nlength ndropn-nnth nnth-nmap*)
have *136*: $\bigwedge j.\ (Suc\ j) \leq nlength\ ls - enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))\ (Suc\ j) = nnth\ ls\ (n{+}\ (Suc\ j)) - nnth\ ls\ n$
using *135* **by** *blast*
have *137*: $\bigwedge j.\ (Suc\ j) \leq nlength\ ls - enat\ n \Longrightarrow$
$nnth\ ls\ (n{+}\ j) - nnth\ ls\ n <\ \ nnth\ ls\ (n{+}\ (Suc\ j)) - nnth\ ls\ n$
by (*metis 13 Suc-ile-eq diff-less-mono le-add1 le-add-same-cancel1 ndropn-nlength ndropn-nnth nidx-expand nidx-less-eq order-less-imp-le*)
have *14*: $nidx\ \ (\ ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))$
by (*metis 134 135 137 Suc-ile-eq dual-order.strict-iff-order nidx-expand*)
have *15*: $g\ \ (pfilt\ \sigma\ (ntaken\ n\ ls))$
by (*metis 10 9 pfilt-nlength pfilt-ntaken*)
have *16*: $h\ (pfilt\ \sigma\ (ndropn\ n\ ls))$
by (*metis 8 pfilt-ndropn pfilt-nlength*)
have *17*: $g\ \ (pfilt\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls))$
by (*metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx*)
have *170*: $nmap\ (\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x))\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ (ndropn\ n\ ls)) =$
$nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls)$
using *nellist.map-comp* **by** *blast*
have *171*: $nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls) =$
$nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)$
proof −
have *172*: $nlength\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls)) =$
$nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))$
by *auto*
have *173*: $\bigwedge j.\ j{\leq}\ nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)) \Longrightarrow$
$nnth\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls))\ j =$
$nnth\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))\ j$
by *auto*
(*metis 13 131 134 le-add1 le-add-diff-inverse le-add-same-cancel1 ndropn-nnth nidx-less-eq nlength-nmap*)
show *?thesis*
by (*metis 172 173 nellist-eq-nnth-eq*)
qed
have *171*: $(pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))) =$
$(pfilt\ \sigma\ (ndropn\ n\ ls))$
using *pfilt-nmap*[*of* $(ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x - (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))$]
pfilt-nmap[*of* $\sigma\ (ndropn\ n\ ls)$] *170 171* **by** *force*

have *18*: *h* (*pfilt* (*ndropn* (*nnth ls n*) *σ*) ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*))))
using *16 171* **by** *auto*
have *19*: *powerinterval f* (*ntaken* (*nnth ls n*) *σ*) (*ntaken n ls*)
by (*metis 3 9 nfinite-conv-nlength-enat pfilt-nlength powerinterval-splita*)
have *20*: *powerinterval f* (*ndropn* (*nnth ls n*) *σ*) ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
using *powerinterval-split*[*of ls n σ f*]
by (*metis 3 9 pfilt-nlength*)
have *21*: (*nnth ls n*) ≤ *nlength σ*
by (*metis 3 9 enat-ord-simps*(*2*) *nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast order.order-iff-strict pfilt-nlength the-enat.simps*)
have *22*: *nnth* (*ntaken n ls*) (*the-enat*(*nlength* (*ntaken n ls*))) = (*nnth ls n*)
by (*metis 9 min-def ntaken-nlength ntaken-nnth pfilt-nlength the-enat.simps*)
have *222*: *nlast* (*ntaken n ls*) = (*nnth ls n*)
by (*simp add*: *ntaken-nlast*)
have *23*: *nnth* ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*the-enat*(*nlength* ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*))))) =
(*the-enat*(*nlength σ*)) −(*nnth ls n*)
by (*metis 171 222 3 9 enat-le-plus-same*(*2*) *enat-ord-simps*(*3*) *enat-the-enat gen-nlength-def ndropn-nfirst nfinite-ndropn-a nfinite-ntaken nfuse-ntaken-ndropn nlast-nfuse nlength-code nnth-nlast nnth-nmap pfilt-nlength*)
have *223*: *nlast* ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*))) = (*the-enat*(*nlength σ*)) −(*nnth ls n*)
by (*metis 23 3 nfinite-ndropn-a nfinite-nmap nnth-nlast*)
have *224*: (*nmap* (*λx*. *x*+ (*nnth ls n*)) ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) = *ndropn n ls*
proof −
have *2241*: *nlength* (*nmap* (*λx*. *x*+ (*nnth ls n*)) ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) =
nlength (*ndropn n ls*)
by *auto*
have *2242*: ⋀*j*. *j*≤*nlength* (*ndropn n ls*) ⟶
nnth (*nmap* (*λx*. *x*+ (*nnth ls n*)) ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *j* =
nnth (*ndropn n ls*) *j*
by (*metis 13 eq-imp-le le-add-diff-inverse2 nidx-gr-first nlength-nmap nnth-nmap not-gr-zero order-less-imp-le*)
show *?thesis* **using** *2241 2242 nellist-eq-nnth-eq*
by *metis*
qed
have *24*: *ls* = *nfuse* (*ntaken n ls*)
(*nmap* (*λx*. *x*+ (*nnth ls n*)) ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*))))
using *nfuse-ntaken-ndropn*
by (*metis 224 9 pfilt-nlength*)
have *241*: *nfinite* (*ntaken n ls*)
by *simp*
have *242*: *nfinite* (*ntaken* (*nnth ls n*) *σ*)
by *simp*
have *243*: *nlast* (*ntaken n ls*) = (*the-enat*(*nlength* (*ntaken* (*nnth ls n*) *σ*)))
by (*simp add*: *21 222*)
have *25*: (∃ *ls1*. *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nfinite* (*ntaken* (*nnth ls n*) *σ*) ∧
nlast ls1 = (*the-enat*(*nlength* (*ntaken* (*nnth ls n*) *σ*))) ∧
powerinterval f (*ntaken* (*nnth ls n*) *σ*) *ls1* ∧ *g* (*pfilt* (*ntaken* (*nnth ls n*) *σ*) *ls1*))
using *12 17 19 241 242 243* **by** *blast*

have *251*: *nfinite* (*ndropn* (*nnth ls n*) σ)
by (*simp add*: *3*)
have *252*: *nlast* (*nmap* (λx. x − *nnth ls n*) (*ndropn n ls*)) = *the-enat*(*nlength* (*ndropn* (*nnth ls n*) σ))
by (*metis 223 3 idiff-enat-enat ndropn-nlength nfinite-nlength-enat the-enat.simps*)
have *26*: ($\exists$ *ls2*. *nidx ls2* $\wedge$ *nnth ls2 0* = *0* $\wedge$ *nfinite ls2* $\wedge$ *nfinite* (*ndropn* (*nnth ls n*) σ) $\wedge$
nlast ls2 = *the-enat*(*nlength* (*ndropn* (*nnth ls n*) σ)) $\wedge$
powerinterval f (*ndropn* (*nnth ls n*) σ) *ls2* $\wedge$ *h* (*pfilt* (*ndropn* (*nnth ls n*) σ) *ls2*))
by (*metis 131 132 14 18 20 252 3 nfinite-ndropn-a nfinite-nmap*)
have *27*: (*ntaken* (*nnth ls n*) σ) $\models$ *f fproj g*
by (*metis 25 fprojection-d-def*)
have *28*: (*ndropn* (*nnth ls n*) σ) $\models$ *f fproj h*
by (*metis 26 fprojection-d-def nfinite-nlength-enat the-enat.simps*)
show *?thesis*
using *21 27 28 chop-defs* **by** *auto*
qed

lemma *OPJ4semchaina*:
assumes (σ $\models$ *f oproj* ((*g* $\wedge$ *finite*);*h*))
shows (σ $\models$ (*f fproj g*) ; (*f oproj h*))
proof −
have *1*: (σ $\models$ *f oproj* ((*g* $\wedge$ *finite*);*h*))
using *assms* **by** *auto*
have *2*: ($\exists$ *ls*. *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))))
using *assms* **unfolding** *chop-defs oprojection-d-def finite-defs*
by *auto*
obtain *ls* **where** *3*: *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *2* **by** *auto*
have *4*: *nidx ls* $\wedge$ *nnth ls 0* = *0*
using *3* **by** *auto*
have *5*: *powerinterval f* σ *ls*
using *3* **by** *auto*
have *7*: ($\exists$ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *3* **by** *auto*
obtain *n* **where** *8*: *n*≤*nlength* (*pfilt* σ *ls*) $\wedge$ *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))
using *7* **by** *auto*
have *9*: *n*≤*nlength* (*pfilt* σ *ls*)
using *8* **by** *auto*
have *10*: *g* (*ntaken n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *11*: *h* (*ndropn n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *12*: *nidx* (*ntaken n ls*) $\wedge$ *nnth* (*ntaken n ls*) *0* = *0*
using *4 8* **unfolding** *nidx-expand pfilt-nlength* **by** *simp*
(*metis 3 Suc-leD bot-nat-0.extremum min.orderE nfinite-ntaken nle-le ntaken-all ntaken-nnth*)
have *13*: *nidx* (*ndropn n ls*) $\wedge$ *nnth* (*ndropn n ls*) *0* = (*nnth ls n*)
using *4 9* **unfolding** *pfilt-nlength nidx-expand*

using *nidx-expand nidx-ndropn* **by** *auto*
have *131*: $((nmap\ (\lambda x.\ x-\ (nnth\ ls\ n))\ \ (ndropn\ n\ ls))) = ndropn\ n\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ ls)$
using *ndropn-nmap*[*of* $n\ (\lambda x.\ x-\ (nnth\ ls\ n))\ ls$] **by** *presburger*
have *132*: $nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ ls))\ 0 = 0$
by (*metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le*)
have *133*: $\bigwedge j.\ enat\ j \leq nlength\ ls \Longrightarrow nnth\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ ls)\ j = nnth\ ls\ j\ -\ nnth\ ls\ n$
using *nnth-nmap* **by** *blast*
have *134*: $nlength\ (ndropn\ n\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ ls)) = nlength\ ls\ -\ enat\ n$
by *simp*
have *135*: $\bigwedge j.\ j \leq nlength\ ls\ -\ enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ ls))\ j = nnth\ ls\ (n+\ j)\ -\ nnth\ ls\ n$
by (*metis 131 ndropn-nlength ndropn-nnth nnth-nmap*)
have *136*: $\bigwedge j.\ (Suc\ j) \leq nlength\ ls\ -\ enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ ls))\ (Suc\ j) = nnth\ ls\ (n+\ (Suc\ j))\ -\ nnth\ ls\ n$
using *135* **by** *blast*
have *137*: $\bigwedge j.\ (Suc\ j) \leq nlength\ ls\ -\ enat\ n \Longrightarrow$
$nnth\ ls\ (n+\ j)\ -\ nnth\ ls\ n <\ \ nnth\ ls\ (n+\ (Suc\ j))\ -\ nnth\ ls\ n$
by (*metis 13 Suc-ile-eq diff-less-mono le-add1 le-add-same-cancel1 ndropn-nlength ndropn-nnth nidx-expand nidx-less-eq order-less-imp-le*)
have *14*: $nidx\ \ (\ ndropn\ n\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ ls))$
by (*metis 134 135 137 Suc-ile-eq dual-order.order-iff-strict nidx-expand*)
have *15*: $g\ \ (pfilt\ \sigma\ (ntaken\ n\ ls))$
by (*metis 10 9 pfilt-nlength pfilt-ntaken*)
have *16*: $h\ (pfilt\ \sigma\ (ndropn\ n\ ls))$
by (*metis 8 pfilt-ndropn pfilt-nlength*)
have *17*: $g\ \ (pfilt\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls))$
by (*metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx*)
have *170*: $nmap\ (\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n\ +\ x))\ (nmap\ (\lambda x.\ x\ -\ nnth\ ls\ n)\ (ndropn\ n\ ls)) =$
$nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n\ +\ x)) \circ (\lambda x.\ x\ -\ nnth\ ls\ n))\ (ndropn\ n\ ls)$
using *nellist.map-comp* **by** *blast*
have *171*: $nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n\ +\ x)) \circ (\lambda x.\ x\ -\ nnth\ ls\ n))\ (ndropn\ n\ ls) =$
$nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)$
proof −
have *172*: $nlength\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n\ +\ x)) \circ (\lambda x.\ x\ -\ nnth\ ls\ n))\ (ndropn\ n\ ls)) =$
$nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))$
by *auto*
have *173*: $\bigwedge j.\ j\leq nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)) \Longrightarrow$
$nnth\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n\ +\ x)) \circ (\lambda x.\ x\ -\ nnth\ ls\ n))\ (ndropn\ n\ ls))\ j =$
$nnth\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))\ j$
by *auto*
(*metis 13 131 134 le-add1 le-add-diff-inverse le-add-same-cancel1 ndropn-nnth nidx-less-eq nlength-nmap*)
show *?thesis*
by (*metis 172 173 nellist-eq-nnth-eq*)
qed
have *171*: $(pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x-\ (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))) =$
$(pfilt\ \sigma\ (ndropn\ n\ ls))$
using *pfilt-nmap*[*of* $(ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x-\ (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))$]
pfilt-nmap[*of* $\sigma\ (ndropn\ n\ ls)$] *170 171* **by** *force*
have *18*: $h\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x-\ (nnth\ ls\ n))\ \ (ndropn\ n\ ls))))$

using *16 171* **by** *auto*
have *19*: *powerinterval f* (*ntaken* (*nnth ls n*) *σ*) (*ntaken n ls*)
by (*metis 3 8 pfilt-nlength powerinterval-splita-alt*)
have *20*: *powerinterval f* (*ndropn* (*nnth ls n*) *σ*) ((*nmap* (*λx. x− *(*nnth ls n*)) (*ndropn n ls*)))
using *powerinterval-split*[*of ls n σ f*]
by (*metis 3 8 pfilt-nlength powerinterval-splitb-alt*)
have *21*: (*nnth ls n*) ≤ *nlength σ*
by (*metis 3 linorder-le-cases nfinite-ntaken ntaken-all*)
have *22*: *nnth* (*ntaken n ls*) (*the-enat*(*nlength* (*ntaken n ls*))) = (*nnth ls n*)
by (*metis 9 min-def ntaken-nlength ntaken-nnth pfilt-nlength the-enat.simps*)
have *222*: *nlast* (*ntaken n ls*) = (*nnth ls n*)
by (*simp add*: *ntaken-nlast*)
have *224*: (*nmap* (*λx. x+ *(*nnth ls n*)) ((*nmap* (*λx. x− *(*nnth ls n*)) (*ndropn n ls*)))) = *ndropn n ls*
proof −
have *2241*: *nlength* (*nmap* (*λx. x+ *(*nnth ls n*)) ((*nmap* (*λx. x− *(*nnth ls n*)) (*ndropn n ls*)))) =
nlength (*ndropn n ls*)
by *auto*
have *2242*: ⋀*j*. *j*≤*nlength* (*ndropn n ls*) ⟶
nnth (*nmap* (*λx. x+ *(*nnth ls n*)) ((*nmap* (*λx. x− *(*nnth ls n*)) (*ndropn n ls*)))) *j* =
nnth (*ndropn n ls*) *j*
by (*metis 13 132 9 diff-is-0-eq le-add-diff-inverse2 nidx-gr-first nlength-nmap nnth-nmap*
nnth-zero-ndropn not-gr-zero order-less-imp-le pfilt-nlength)
show *?thesis* **using** *2241 2242 nellist-eq-nnth-eq*
by *metis*
qed
have *24*: *ls* = *nfuse* (*ntaken n ls*)
(*nmap* (*λx. x+ *(*nnth ls n*)) ((*nmap* (*λx. x− *(*nnth ls n*)) (*ndropn n ls*))))
using *nfuse-ntaken-ndropn*
by (*metis 224 9 pfilt-nlength*)
have *25*: (∃ *ls1*. *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nfinite* (*ntaken* (*nnth ls n*) *σ*) ∧
nlast ls1 = (*the-enat*(*nlength* (*ntaken* (*nnth ls n*) *σ*))) ∧
powerinterval f (*ntaken* (*nnth ls n*) *σ*) *ls1* ∧ *g* (*pfilt* (*ntaken* (*nnth ls n*) *σ*) *ls1*))
using *12 17 19 21 222 3*
by (*metis min.orderE nfinite-ntaken ntaken-nlength the-enat.simps*)
have *26*: (∃ *ls2*. *nidx ls2* ∧ *nnth ls2 0* = *0* ∧ ¬*nfinite ls2* ∧ ¬*nfinite* (*ndropn* (*nnth ls n*) *σ*) ∧
powerinterval f (*ndropn* (*nnth ls n*) *σ*) *ls2* ∧ *h* (*pfilt* (*ndropn* (*nnth ls n*) *σ*) *ls2*))
using *assms 14 18 20 3 132 131* **by** (*metis nfinite-ndropn nfinite-nmap*)
have *27*: (*ntaken* (*nnth ls n*) *σ*) ⊨ *f fproj g*
by (*metis 25 fprojection-d-def*)
have *28*: (*ndropn* (*nnth ls n*) *σ*) ⊨ *f oproj h*
by (*metis 26 oprojection-d-def*)
show *?thesis*
using *21 27 28 chop-defs* **by** *auto*
qed

lemma *PJ4semchainb*:
assumes (*σ* ⊨ (*f fproj g*) ; (*f fproj h*))
shows (*σ* ⊨ *f fproj* (*g*;*h*))
proof −
have *1*: (∃ *n*≤*nlength σ*.

$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ n\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ n\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ n\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ n\ \sigma)\ ls)) \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ndropn\ n\ \sigma) \wedge$
$nlast\ ls = (the\text{-}enat(nlength\ (ndropn\ n\ \sigma))) \wedge$
$powerinterval\ f\ (ndropn\ n\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ n\ \sigma)\ ls)))$
using *assms* **unfolding** *chop-defs fprojection-d-def* **by** *simp blast*
obtain *cp* **where** *2*: $cp \leq nlength\ \sigma \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls)) \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
$nlast\ ls = (the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *1* **by** *auto*
have *3*: $cp \leq nlength\ \sigma$
using *2* **by** *auto*
have *4*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls1* **where** *5*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0 \wedge nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls1\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1 \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *4* **by** *auto*
have *6*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0$
using *5* **by** *auto*
have *61*: $nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma)$
using *5* **by** *auto*
have *7*: $nlast\ ls1\ = the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))$
using *5* **by** *auto*
have *8*: $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1$
using *5* **by** *auto*
have *9*: $g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *5* **by** *auto*
have *10*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
$nlast\ ls = (the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls2* **where** *11*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0 \wedge nfinite\ ls2 \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
$nlast\ ls2 = (the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2 \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using *10* **by** *auto*
have *12*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0$
using *11* **by** *auto*
have *13*: $nlast\ ls2\ = the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))$
using *11* **by** *auto*
have *14*: $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2$
using *11* **by** *auto*
have *15*: $h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$

using *11* **by** *auto*
have *150*: *nlast ls1 = cp*
by (*simp add*: *2 7*)
have *151*: *nfirst* (*nmap* (λ*x*. *x*+ *cp*) *ls2*) = *cp*
by (*metis 11 NNil-eq-nmap-conv nlast-NNil ntake-eq-NNil-iff ntake-nmap ntaken-0 ntaken-nlast plus-nat.add-0*)
have *152*: *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) *0* = *0*
by (*metis 150 151 5 nfuse-nnth zero-enat-def zero-le*)
have *153*: *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nlast ls1* = *cp*
by (*simp add*: *150 5*)
have *154*: *nidx ls2* ∧ *nnth ls2 0* = *0* ∧ *nfinite ls2* ∧ *nfinite* σ ∧ *nlast ls2* = *the-enat* (*nlength* σ) − *cp*
by (*metis 11 idiff-enat-enat ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn the-enat.simps*)
have *16*: *nidx* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) ∧ *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) *0* = *0*
using *nidx-nfuse-nidx*[*of ls1 ls2 cp* σ] *153 154 3* **by** *fastforce*
have *17*: *nlast ls1* = *nfirst* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)
by (*simp add*: *150 151*)
have *18*: *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) (*the-enat*(*nlength ls1*)) = *cp*
by (*metis 151 17 5 ntaken-nfuse ntaken-nlast*)
have *19*: *nlast* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) = (*the-enat* (*nlength* σ))
by (*metis 154 17 3 5 diff-add enat.distinct*(*2*) *enat-ord-simps*(*1*) *enat-the-enat nfinite-conv-nlength-enat nlast-nfuse nlast-nmap*)
have *20*: *powerinterval f* σ (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
using *powerinterval-nfuse*[*of ls1* (*ls2*) *cp* σ *f*] *11 150 154 3 5* **by** *fastforce*
have *21*: σ = *nfuse* (*ntaken cp* σ) (*ndropn cp* σ)
by (*simp add*: *2 nfuse-ntaken-ndropn*)
have *22*: *nnth* ((*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))) (*the-enat*(*nlength ls1*)) = *cp*
using *18* **by** *blast*
have *23*: (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)))) =
(*pfilt* (*ntaken cp* σ) *ls1*)
by (*metis 151 17 2 5 enat.distinct*(*2*) *enat-le-plus-same*(*1*) *enat-the-enat nfinite-conv-nlength-enat nfuse-nlength ntaken-nfuse pfilt-ntaken pfilt-ntaken-nidx*)
have *24*: *g* (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))))
by (*simp add*: *23 9*)
have *25*: (*ndropn* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)))) =
(*pfilt* (*ndropn cp* σ) *ls2*)
using *pfilt-ndropn*[*of* (*the-enat*(*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) σ]
pfilt-ndropn-nidx[*of cp* σ *ls2*]
by (*metis 12 17 23 3 5 enat-ord-code*(*4*) *enat-the-enat min-def ndropn-nfuse nle-le ntaken-nlength order-less-imp-le pfilt-nlength*)
have *26*: *nlength ls1* ≤ *nlength* (*pfilt* σ (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)))
by (*simp add*: *nfuse-nlength pfilt-nlength*)
have *27*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite* σ ∧
nlast ls = *the-enat*(*nlength* σ) ∧
powerinterval f σ *ls* ∧
(∃ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) ∧ *h* (*ndropn n* (*pfilt* σ *ls*))))
using *11 16 19 20 24 25 26*
by (*metis 5 nfinite-ndropn nfinite-nlength-enat nfinite-nmap pfilt-nmap the-enat.simps*)
show *?thesis* **unfolding** *fprojection-d-def* **using** *chop-nfuse nfuse-ntaken-ndropn 27*
by (*metis ndropn-nfirst nfinite-conv-nlength-enat nfinite-ntaken ntaken-nlast*)
qed

lemma *OPJ4semchainb*:
assumes $(\sigma \models (f \ \ fproj \ \ g) \ ; (f \ oproj \ h))$
shows $(\sigma \models f \ oproj \ ((g \wedge finite);h))$
proof −
have *1*: $(\exists\, n{\leq}nlength\ \sigma.$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ n\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ n\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ n\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ n\ \sigma)\ ls)) \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ (ndropn\ n\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ n\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ n\ \sigma)\ ls)))$
using *assms* **unfolding** *chop-defs fprojection-d-def oprojection-d-def*
by *simp blast*
obtain *cp* **where** *2*: $cp{\leq}nlength\ \sigma \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls)) \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ (ndropn\ cp\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *1* **by** *auto*
have *3*: $cp{\leq}nlength\ \sigma$
using *2* **by** *auto*
have *4*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls1* **where** *5*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0 \wedge nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls1\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1 \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *4* **by** *auto*
have *6*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0$
using *5* **by** *auto*
have *61*: $nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma)$
using *5* **by** *auto*
have *7*: $nlast\ ls1\ = the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))$
using *5* **by** *auto*
have *8*: $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1$
using *5* **by** *auto*
have *9*: $g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *5* **by** *auto*
have *10*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ (ndropn\ cp\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls2* **where** *11*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0 \wedge \neg nfinite\ ls2 \wedge \neg nfinite\ (ndropn\ cp\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2 \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using *10* **by** *auto*
have *12*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0$
using *11* **by** *auto*
have *14*: $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2$

using *11* **by** *auto*
have *15*: *h* (*pfilt* (*ndropn cp* σ) *ls2*)
using *11* **by** *auto*
have *150*: *nlast ls1* = *cp*
by (*simp add*: *2 7*)
have *151*: *nfirst* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*) = *cp*
by (*metis 11 NNil-eq-nmap-conv nlast-NNil ntake-eq-NNil-iff ntake-nmap ntaken-0 ntaken-nlast plus-nat.add-0*)
have *152*: *nnth* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) *0* = *0*
by (*metis 150 151 5 nfuse-nnth zero-enat-def zero-le*)
have *153*: *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nlast ls1* = *cp*
by (*simp add*: *150 5*)
have *154*: *nidx ls2* ∧ *nnth ls2 0* = *0* ∧ ¬*nfinite ls2* ∧ ¬*nfinite* σ
using *11 nfinite-ndropn-a* **by** *blast*
have *16*: *nidx* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) ∧ *nnth* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) *0* = *0*
using *nidx-nfuse-nidx-infinite*[*of ls1 ls2 cp* σ] *150 154 5* **by** *fastforce*
have *17*: *nlast ls1* = *nfirst* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)
by (*simp add*: *150 151*)
have *18*: *nnth* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) (*the-enat*(*nlength ls1*)) = *cp*
by (*metis 150 17 5 ntaken-nfuse ntaken-nlast*)
have *20*: *powerinterval f* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*))
using *powerinterval-nfuse-alt*[*of ls1* (*ls2*) *cp* σ *f*]
using *14 154 3 5* **by** *fastforce*
have *21*: σ = *nfuse* (*ntaken cp* σ) (*ndropn cp* σ)
by (*simp add*: *2 nfuse-ntaken-ndropn*)
have *22*: *nnth* ((*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*))) (*the-enat*(*nlength ls1*)) = *cp*
using *18* **by** *blast*
have *23*: (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)))) =
(*pfilt* (*ntaken cp* σ) *ls1*)
by (*metis 150 17 2 5 enat.distinct*(*2*) *enat-le-plus-same*(*1*) *enat-the-enat nfinite-conv-nlength-enat nfuse-nlength ntaken-nfuse pfilt-ntaken pfilt-ntaken-nidx*)
have *24*: *g* (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*))))
by (*simp add*: *23 9*)
have *25*: (*ndropn* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)))) =
(*pfilt* (*ndropn cp* σ) *ls2*)
using *pfilt-ndropn*[*of* (*the-enat*(*nlength ls1*)) (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) σ]
pfilt-ndropn-nidx[*of cp* σ *ls2*]
by (*metis 12 17 2 23 5 enat-ord-code*(*4*) *enat-the-enat min-def ndropn-nfuse nle-le ntaken-nlength order-less-imp-le pfilt-nlength*)
have *26*: *nlength ls1* ≤ *nlength* (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)))
by (*simp add*: *nfuse-nlength pfilt-nlength*)
have *27*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite* σ ∧
powerinterval f σ *ls* ∧
(∃ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) ∧ *h* (*ndropn n* (*pfilt* σ *ls*))))
using *11 16 20 24 25 26*
by (*metis 154 5 nfinite-nlength-enat nfinite-nmap nfuse-nfinite the-enat.simps*)
show *?thesis* **unfolding** *fprojection-d-def oprojection-d-def chop-nfuse nfuse-ntaken-ndropn finite-defs* **by** *simp*
(*metis 27 ndropn-nfirst nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast*)
qed

lemma *PJ4sem*:
$(\sigma \models f\ \ fproj\ \ (g;h) = (f\ \ fproj\ \ g)\ ;\ (f\ \ fproj\ \ h))$
using *PJ4semchaina PJ4semchainb unl-lift2* **by** *blast*

lemma *OPJ4sem*:
$(\sigma \models f\ oproj\ ((g \wedge finite);h) = (f\ \ fproj\ \ g)\ ;\ (f\ oproj\ h))$
using *OPJ4semchaina OPJ4semchainb unl-lift2* **by** *blast*

PJ5

lemma *PJ5sem*:
$(\sigma \models f\ \ fproj\ \ init(g) \longrightarrow init(g))$
by (*simp add*: *fprojection-d-def init-defs*)
(*metis ndropn-0 ndropn-nfirst ntaken-0 pfilt-code(1) pfilt-ntaken zero-enat-def zero-le*)

lemma *OPJ5sem*:
$(\sigma \models f\ oproj\ init(g) \longrightarrow init(g))$
by (*simp add*: *oprojection-d-def init-defs*)
(*metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le*)

PJ6

lemma *PJ6help1*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
nfinite ls
nfinite σ
$nlast\ ls = the\text{-}enat(nlength\ \sigma)$
shows $(\forall\ i.\ 0 \leq i \wedge i < nlength\ ls \longrightarrow nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
$= (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i))$
proof
fix *i*
show $0 \leq i \wedge i < nlength\ ls \longrightarrow$
$nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = nnth\ ls\ (Suc\ i) - nnth\ ls\ i$
using *assms nsubn-nlength*[*of* σ] **unfolding** *nidx-expand*
by (*metis assms(1) dual-order.order-iff-strict eSuc-enat enat-minus-mono1 enat-ord-simps(1)*
idiff-enat-enat ileI1 min.orderE nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast
the-enat.simps)
qed

lemma *OPJ6help1*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
$\neg nfinite\ ls$
$\neg nfinite\ \sigma$
shows $(\forall\ i.\ 0 \leq i \wedge i < nlength\ ls \longrightarrow nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
$= (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i))$
proof
fix *i*
show $0 \leq i \wedge i < nlength\ ls \longrightarrow$

nlength (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) = *nnth ls* (*Suc i*) − *nnth ls i*
using *assms* **unfolding** *nidx-expand nsubn-def1* **by** (*simp add*: *nfinite-conv-nlength-enat*)
qed

lemma *PJ6help2*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength* σ) ∧
(∀ *i*<*nlength ls*. *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*)
shows (∀ *i*≤*nlength ls*. *nnth ls i* = *i*)
proof
fix *i*
show *i*≤*nlength ls* ⟶ *nnth ls i* = *i*
proof
(*induct i*)
case *0*
then show *?case* **using** *assms* **by** *blast*
next
case (*Suc i*)
then show *?case*
by (*metis One-nat-def Suc-ile-eq assms*(*1*) *assms*(*5*) *diff-add nidx-expand order-less-imp-le plus-1-eq-Suc*)
qed
qed

lemma *OPJ6help2*:
assumes *nidx ls*
nnth ls 0 = 0
¬*nfinite ls*
¬*nfinite* σ
(∀ *i*<*nlength ls*. *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*)
shows (∀ *i*≤*nlength ls*. *nnth ls i* = *i*)
proof
fix *i*
show *i*≤*nlength ls* ⟶ *nnth ls i* = *i*
proof
(*induct i*)
case *0*
then show *?case* **using** *assms* **by** *blast*
next
case (*Suc i*)
then show *?case*
by (*metis One-nat-def Suc-ile-eq assms*(*1*) *assms*(*5*) *diff-add nidx-expand order-less-imp-le plus-1-eq-Suc*)
qed
qed

lemma *PJ6help3*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ i≤nlength ls. nnth ls i = i)
shows *(∀ i<nlength ls. nnth ls (Suc i) − nnth ls i = Suc 0)*
proof
fix *i*
show *i<nlength ls ⟶ nnth ls (Suc i) − nnth ls i = Suc 0*
by (*metis One-nat-def add-diff-cancel-right′ assms(6) eSuc-enat ileI1 order-less-imp-le plus-1-eq-Suc*)
qed

lemma *OPJ6help3*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ i≤nlength ls. nnth ls i = i)
shows *(∀ i<nlength ls. nnth ls (Suc i) − nnth ls i = Suc 0)*
proof
fix *i*
show *i<nlength ls ⟶ nnth ls (Suc i) − nnth ls i = Suc 0*
by (*metis One-nat-def add-diff-cancel-right′ assms(5) eSuc-enat ileI1 order-less-imp-le plus-1-eq-Suc*)
qed

lemma *PJ6help4*:
assumes *nfinite σ*
shows *(∃ ls. nidx ls ∧ nnth ls 0 = 0 ∧ nfinite ls ∧ nfinite σ ∧*
ls = ntaken (the-enat (nlength σ)) (niterates Suc 0) ∧
nlast ls = the-enat(nlength σ) ∧
(∀ i≤nlength ls. nnth ls i = i) ∧
(∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ (i)) ∧ σ1 = σ))

proof −
have *1*: *nidx (ntaken (the-enat(nlength σ)) (niterates Suc 0))*
using *nidx-ntaken-niterates-Suc* **by** *presburger*
have *2*: *nnth (ntaken (the-enat (nlength σ)) (niterates Suc 0)) 0 = 0*
by (*simp add*: *ntaken-nnth*)
have *3*: *nlast (ntaken (the-enat (nlength σ)) (niterates Suc 0)) = the-enat(nlength σ)*
by (*simp add*: *ntaken-nlast*)
have *4*: *(∀ i≤nlength (ntaken (the-enat (nlength σ)) (niterates Suc 0)).*
nnth (ntaken (the-enat (nlength σ)) (niterates Suc 0)) i = i)
by (*simp add*: *ntaken-nnth*)

have *5*: $(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ (ntaken\ (the\text{-}enat\ (nlength\ \sigma))\ (niterates\ Suc\ 0)) \wedge$
$(\forall i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ i) \wedge \sigma 1 = \sigma)$
using *assms* **by** (*simp add*: *enat-the-enat nfinite-nlength-enat*)
show *?thesis*
using *1 2 3 4 5 assms nfinite-ntaken* **by** *blast*
qed

lemma *OPJ6help4*:
assumes $\neg nfinite\ \sigma$
shows $(\exists ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$ls = (niterates\ Suc\ 0)\ \wedge$
$(\forall i{\leq}nlength\ ls.\ nnth\ ls\ i = i) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge \sigma 1 = \sigma))$

proof −
have *1*: $nidx\ ((niterates\ Suc\ 0))$
using *nidx-expand nnth-niterates-Suc* **by** *presburger*
have *2*: $nnth\ (\ (niterates\ Suc\ 0))\ 0 = 0$
by (*simp*)
have *4*: $(\forall i{\leq}nlength\ (\ (niterates\ Suc\ 0)).\ nnth\ (\ (niterates\ Suc\ 0))\ i = i)$
by (*simp*)
have *5*: $(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ (\ (niterates\ Suc\ 0)) \wedge$
$(\forall i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ i) \wedge \sigma 1 = \sigma)$
using *assms* **by** (*simp add*: *nfinite-conv-nlength-enat*)
show *?thesis*
using *1 2 4 5 assms* **by** *blast*
qed

lemma *PJ6help5*:
assumes *nfinite* σ
shows $(\exists ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ \ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall i{<}nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (\ nnth\ ls\ i)) \wedge g\ \sigma 1))$
$= g\ \sigma$
proof −
have $(\exists ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ \ ls\ = the\text{-}enat\ (nlength\ \sigma) \wedge$
$(\forall i{<}nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$
$=$
$(\exists ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall i{<}nlength\ ls.\ nnth\ ls\ (Suc\ i) - nnth\ ls\ i = Suc\ 0) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (\ nnth\ ls\ i)) \wedge g\ \sigma 1))$
using *PJ6help1*[*of* - σ] *assms* **by** *auto*
also have $\ldots =$
$(\exists ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$

```
        nlast  ls  = the-enat(nlength σ) ∧
       (∀ i≤nlength ls. nnth ls i = i) ∧
       (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ (nnth ls i)) ∧ g σ1))

     using PJ6help2 PJ6help3 by blast
also have  ... =
          (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ nfinite ls ∧ nfinite σ ∧
        nlast ls  = the-enat(nlength σ) ∧
       (∀ i≤nlength ls. nnth ls i = i) ∧
       (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ (i)) ∧ g σ1))

    using assms by auto (metis, metis)
also have  ... =
          (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ nfinite ls ∧ nfinite σ ∧
        nlast ls  = the-enat(nlength σ) ∧
       (∀ i≤nlength ls. nnth ls i = i) ∧
       (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ (i)) ∧ σ1 = σ ∧ g σ))
      by auto
      (metis dual-order.refl enat.distinct(2) enat-the-enat nellist-eq-nnth-eq nfinite-nlength-enat
        nnth-nlast)
also have  ... =
          g σ

  using PJ6help4 assms by auto
finally show (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ nfinite ls ∧ nfinite σ ∧
        nlast ls  = the-enat(nlength σ) ∧
       (∀ i<nlength ls. nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) = Suc 0) ∧
       (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ ( nnth ls i)) ∧ g σ1))
  = g σ
          .
qed

lemma OPJ6help5:
assumes ¬nfinite σ
shows (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ ¬nfinite ls ∧ ¬nfinite σ ∧
       (∀ i<nlength ls. nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) = Suc 0) ∧
       (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ ( nnth ls i)) ∧ g σ1))
  = g σ
proof −
 have (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ ¬nfinite ls ∧ ¬nfinite σ ∧
       (∀ i<nlength ls. nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) = Suc 0) ∧
       (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ (nnth ls i)) ∧ g σ1))
     =
     (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ ¬nfinite ls ∧ ¬nfinite σ ∧
       (∀ i<nlength ls. nnth ls (Suc i) − nnth ls i = Suc 0) ∧
       (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i≤nlength σ1. nnth σ1 i = nnth σ ( nnth ls i)) ∧ g σ1))
    using OPJ6help1[of - σ] assms by auto
  also have  ... =
          (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ ¬nfinite ls ∧ ¬nfinite σ ∧
       (∀ i≤nlength ls. nnth ls i = i) ∧
```

$(\exists\,\sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

using *OPJ6help2*[*of* - σ] *OPJ6help3*[*of* - σ] **by** *blast*
also have ... =
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{\leq}nlength\ ls.\ nnth\ ls\ i = i) \wedge$
$(\exists\,\sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge g\ \sigma 1))$

using *assms* **by** *auto* (*metis*, *metis*)
also have ... =
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{\leq}nlength\ ls.\ nnth\ ls\ i = i) \wedge$
$(\exists\,\sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge \sigma 1 = \sigma \wedge g\ \sigma))$

by *auto* (*metis OPJ6help4 nellist-eq-nnth-eq*)
also have ... =
$g\ \sigma$

using *OPJ6help4 assms* **by** *auto*
finally show $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{<}nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\,\sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (\ nnth\ ls\ i)) \wedge g\ \sigma 1))$
$= g\ \sigma$
.
qed

lemma *PJ6sem*:
$(\sigma \models skip\ \ fproj\ \ g = (g \wedge finite))$
proof −
have *1*: $(\sigma \models skip\ \ fproj\ \ g = (g \wedge finite)) =$
$((\exists\, l.\ nidx\ \ l \wedge nnth\ l\ 0 = 0 \wedge nfinite\ l \wedge nfinite\ \sigma \wedge nlast\ l\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i{<}nlength\ l.\ \ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ l)) =$
$(g\ \sigma \wedge nfinite\ \sigma)\ )$
by (*simp add*: *fprojection-d-def powerinterval-def finite-defs*)
have *2*: $(\exists\, ls.\ nidx\ \ ls \wedge\ \ nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i{<}nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ ls)) =$
$(\exists\, ls\ \sigma 1\ .\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ \ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i{<}nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ \ i{\leq}\ nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1)$
using *pfilt-expand*[*of* - σ]
by *auto* (*blast*,*metis*)
have *3*: $(\exists\, ls\ \sigma 1\ .\ nidx\ \ ls \wedge\ \ \ nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ \ ls = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i{<}nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ \ i{\leq}\ nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1) =$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$

$(\forall\, i{<}nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$
by (*simp add*: *skip-defs*)
have *4*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge\ \ nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i{<}nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1)) =$
$((g\ \sigma) \wedge nfinite\ \sigma)$
using *PJ6help5*[*of* σ *g*] **by** *auto*
from *1 2 3 4* **show** *?thesis*
by *simp*
qed

lemma *OPJ6sem*:
$(\sigma \models skip\ oproj\ g = (g \wedge inf))$
proof −
have *1*: $(\sigma \models skip\ oproj\ g = (g \wedge inf)) =$
$((\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{<}nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ ls)) =$
$(g\ \sigma \wedge \neg nfinite\ \sigma)\ )$
by (*simp add*: *oprojection-d-def powerinterval-def infinite-defs*)
have *2*: $(\exists\, ls.\ nidx\ \ ls \wedge\ \ nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{<}nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ ls)) =$
$(\exists\, ls\ \sigma 1\ .\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{<}nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ \ i{\leq}\ nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1)$
using *pfilt-expand*[*of* - σ] **by** *auto* (*blast*,*metis*)
have *3*: $(\exists\, ls\ \sigma 1\ .\ nidx\ \ ls \wedge\ \ \ nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{<}nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ \ i{\leq}\ nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1) =$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{<}nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$
by (*simp add*: *skip-defs*)
have *4*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i{<}nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i{\leq}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1)) =$
$((g\ \sigma) \wedge \neg nfinite\ \sigma)$
using *OPJ6help5*[*of* σ *g*] **by** *auto*
from *1 2 3 4* **show** *?thesis*
by *simp*
qed

PJ7

lemma *PJemptyImp*:
assumes *nlength* $\sigma = 0$

shows $(\sigma \models (f \ \ fproj \ \ g) = g)$
using *assms*
by (*auto simp add*: *fprojection-d-def* *powerinterval-def nsubn-def1*)
(*metis i0-less less-numeral-extra*(*3*) *ndropn-0 ndropn-nlast nfinite-conv-nlength-enat*
nidx-less-last-1 nnth-nlast pfilt-code(*1*) *the-enat.simps the-enat-0*,
metis PJ6help4 ndropn-0 ndropn-nlast nfinite-ntaken not-less-zero ntaken-all ntaken-nlast
pfilt-code(*1*) *zero-le*)

lemma *PJ7empty*:
assumes *nlength* $\sigma = 0$
shows $(\sigma \models f \ \ fproj \ \ (g \ \ fproj \ \ h) = (f \ \ fproj \ \ g) \ \ fproj \ \ h)$
proof −
have *1*: $(\sigma \models f \ \ fproj \ \ (g \ \ fproj \ \ h) = (g \ \ fproj \ \ h))$
using *PJemptyImp assms* **by** *blast*
have *2*: $(\sigma \models (g \ \ fproj \ \ h) = h)$
using *PJemptyImp assms* **by** *blast*
have *3*: $(\sigma \models (f \ \ fproj \ \ g) \ \ fproj \ \ h = h)$
using *PJemptyImp assms* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *simp*
qed

lemma *PJ7helpchain1a-help-1*:
assumes *nidx* *ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength* σ)
$(\forall \ i{<}$*nlength ls*. (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models f \ \ fproj \ \ g)$
nlength $\sigma > 0$
shows *nidx* (*nfusecat* ((*lcppl f g* σ *ls*))) $\wedge$ *nnth* (*nfusecat* ((*lcppl f g* σ *ls*))) *0* = *0*
proof −
have *0*: *nlength* $\sigma > 0 \longrightarrow$ *nlength ls* >*0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *01*: *nfirst ls* = *0*
using *assms*
by (*metis ndropn-0 ndropn-nfirst*)
have *02*: *nlastnfirst* (*lcppl f g* σ *ls*)
using *assms lcppl-nfusecat-nlastnfirst*[*of ls* σ *f g*]
by (*metis 0*)
have *1*: *nfirst* (*nfusecat* (*lcppl f g* σ *ls*)) = *0*
using *assms 0* *01* *02 lcppl-nfirst*[*of ls* σ *f g*] **by** (*metis* *nfirst-nfusecat-nfirst*)
from *1 0* **show** *?thesis* **using** *assms lcppl-nfusecat-nidx*[*of ls* σ *f g*]
by (*metis 01* *ntaken-0 ntaken-nlast*)
qed

lemma *PJ7helpchain1a-help-1-alt*:
assumes *nidx* *ls*
nnth ls 0 = *0*
$\neg$*nfinite ls*

```
        ¬nfinite σ
         (∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f fproj g)
         nlength σ > 0
 shows    nidx  (nfusecat ( (lcppl f g σ ls))) ∧ nnth (nfusecat ( (lcppl f g σ ls))) 0 = 0
proof −
 have 0:  nlength σ > 0 ⟶ nlength ls >0
   using assms
   by (metis enat-the-enat nfinite-conv-nlength-enat )
 have 01: nfirst ls = 0
  using assms
  by (metis ndropn-0 ndropn-nfirst)
 have 02: nlastnfirst (lcppl f g σ ls)
   using assms lcppl-nfusecat-nlastnfirst-alt[of ls σ f g]
   by (metis )
 have 1:  nfirst (nfusecat  (lcppl f g σ ls)) = 0
  using assms 0  01  02
    lcppl-nfirst-alt[of ls σ f g ]
    by (metis  nfirst-nfusecat-nfirst )
 from 1 0 show ?thesis using assms lcppl-nfusecat-nidx-alt[of ls σ f g]
 by (metis 01   ntaken-0 ntaken-nlast)
qed

lemma PJ7helpchain1a-help-2:
 assumes nidx  ls
        nnth ls 0 = 0
        nfinite ls
        nfinite σ
         nlast ls = the-enat(nlength σ)
         (∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f fproj g)
         nlength σ > 0
 shows    powerinterval f σ (nfusecat ( (lcppl f g σ ls)))
proof −
 have 1: (∀ i<nlength ls.
           powerinterval f (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
                          (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))
  using assms cppl-fprojection by blast
 have 2: ∀ i<nlength ls.
          ∀ ia<nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )).
           f (nsubn  (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
                    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) ia)
                    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) (Suc ia))
                )
  using 1 by (simp add: powerinterval-def)
 have 3: ∀ i<nlength ls.
          ∀ ia<nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )).
            (nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
                   (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) ia)
                   (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) (Suc ia))
                ) =
           (nsubn σ
```

```
          (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))) ia)
          (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))) (Suc ia))
          )
proof auto
 fix i
 fix ia
 assume a0: enat i < nlength ls
 assume a1: enat ia < nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
 show nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)))
          (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
          (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)) =
      nsubn σ (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia + nnth ls i)
          (nnth (nmap (λx. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
 proof −
  have 300: 0 < nlength σ
   using assms(7) by auto
  have 301: nnth ls i < nnth ls (Suc i)
   by (metis a0 assms(1) eSuc-enat ileI1 nidx-expand)
  have 302: (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f  fproj  g)
    by (simp add: a0 assms(6))
  have 303: the-enat (nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)))) = (nnth ls (Suc i)) − (nnth ls i)
    by (simp add: PJ6help1 a0 assms(1) assms(2) assms(3) assms(4) assms(5))
  have 304: nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) = (nnth ls (Suc i)) − (nnth ls i)
     using  a0 a1 cppl-fprojection[of f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))]
     using 302 303 by presburger
  have 305: (Suc ia) ≤ nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
    by (simp add: Suc-ile-eq a1)
  have 306: nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia ≤
          nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)
    by (metis 302 a1 cppl-fprojection eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)
  have 307: nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))=
          nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
              (the-enat(nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))))
     using 302 cppl-fprojection nnth-nlast by auto
  have 308: nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia) ≤
          nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
       by (metis 302 305 cppl-fprojection nidx-all-le-nlast)
  have 310: enat (nnth ls (Suc i)) ≤ nlength σ
    using a0 assms
     by (metis  dual-order.refl eSuc-enat enat-ord-simps(1) enat-ord-simps(3) enat-the-enat
        ileI1 nfinite-conv-nlength-enat nidx-less-eq nnth-nlast the-enat.simps)
  have 30: nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia) ≤ nnth ls (Suc i) − nnth ls i
     using assms cppl-bounds[of  (nnth ls i) (nnth ls (Suc i)) σ f g Suc ia]
     PJ6help1[of ls σ ]  a0 a1 cppl-fprojection[of f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))]
     by (metis 303 308)
  have 311: nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)))
          (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
          (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)) =
         nsubn σ (nnth ls i + nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
                (nnth ls i + nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia))
```

using *nsubn-nsubn-1*[*of* (*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) (*nnth ls i*) (*nnth ls* (*Suc i*)) *σ*]
using *30 301 306 310 order-less-imp-le* **by** *blast*
have *312*: (*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia* + *nnth ls i*) =
(*nnth ls i* + *nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
by *simp*
have *313*: (*nnth ls i* + *nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) =
(*nnth* (*nmap* (*λx*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
using *305* **by** *force*
show *?thesis*
using *311 312 313* **by** *presburger*
qed
qed
have *4*: ∀ *i*<*nlength ls*.
∀ *ia*<*nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))).
f (*nsubn σ* (*nnth* (*nmap* (*λx*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) *ia*)
(*nnth* (*nmap* (*λx*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
)
using *2 3* **by** *auto*
have *5*: *nlength*(*lcppl f g σ ls*) = *nlength ls* −*1*
using *assms lcppl-nlength lcppl-nlength-zero*
by (*metis epred-0 epred-conv-minus i0-less*)
have *6*: *nlength σ* >*0* ⟶ *nlength ls* >*0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *7*: ∀ *i*< *nlength ls*.
∀ *ia*<*nlength* ((*nnth* (*lcppl f g σ ls*) *i*)).
f (*nsubn σ* (*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *ia*)
(*nnth* (*nnth* (*lcppl f g σ ls*) *i*) (*Suc ia*))
)
using *assms 4 lcppl-nnth* **by** (*metis nlength-nmap*)
have *8*: *nlength σ* >*0* ⟶
(∀ *i*≤ *nlength*(*lcppl f g σ ls*).
∀ *ia*<*nlength* ((*nnth* (*lcppl f g σ ls*) *i*)).
f (*nsubn σ* (*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *ia*)
(*nnth* (*nnth* (*lcppl f g σ ls*) *i*) (*Suc ia*))
))
using *5 6 7 assms* **by** (*metis co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength*)
have *9*: *nlength σ* >*0* ⟶
powerinterval f σ (*nfusecat* ((*lcppl f g σ ls*))) =
(∀ *i*<*nlength* (*nfusecat* (*lcppl f g σ ls*)).
f (*nsubn σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*)) *i*) (*nnth* (*nfusecat* (*lcppl f g σ ls*)) (*Suc i*))))
by (*simp add*: *powerinterval-def*)
have *10*: *nlength σ* >*0* ⟶ *nlastnfirst* (*lcppl f g σ ls*)
using *6 assms lcppl-nfusecat-nlastnfirst*
by (*metis nfinite-conv-nlength-enat*)
have *11*: *nlength σ* >*0* ⟶
(∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)

using *assms*

```
    by (metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat)
  have 110:  all-gr-zero (lcppl f g σ ls)
    using 11 all-gr-zero-nnth-a assms(7) by blast
  have 111:  all-nfinite (lcppl f g σ ls)
    using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite by blast
  have 12: nlength σ >0 ⟶
          (∀ i≤ nlength(lcppl f g σ ls).
            (∀ ia<nlength ((nnth (lcppl f g σ ls) i)).
              f (nsubn σ (nnth (nnth (lcppl f g σ ls) i) ia)
                    (nnth (nnth (lcppl f g σ ls) i) (Suc ia))
                    ))) =
             (∀ j<nlength (nfusecat (lcppl f g σ ls)).
             f (nsubn σ (nnth (nfusecat (lcppl f g σ ls)) j)
                    (nnth (nfusecat (lcppl f g σ ls)) (Suc j))
                    ))

    using  nfusecat-split-nsubn[of  (lcppl f g σ ls) f σ] 6 10 11 110 111 assms
    unfolding nridx-expand
    by (simp add: Suc-ile-eq)
  show ?thesis using 12 8 9 using assms
  by meson
qed

lemma PJ7helpchain1a-help-2-alt:
 assumes nidx  ls
        nnth ls 0 = 0
        ¬nfinite ls
        ¬nfinite σ
         (∀  i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f  fproj  g)
 shows    powerinterval f σ (nfusecat ( (lcppl f g σ ls)))
proof −
 have 1: (∀  i<nlength ls.
            powerinterval f (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
                          (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))
  using assms cppl-fprojection by blast
 have 2: ∀ i<nlength ls.
          ∀ ia<nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )).
           f (nsubn  (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
                    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) ia)
                    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) (Suc ia))
                  )
  using 1 by (simp add: powerinterval-def)
 have 3: ∀ i<nlength ls.
          ∀ ia<nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )).
            (nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
                    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) ia)
                    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) (Suc ia))
                  ) =
           (nsubn σ
                (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))) ia)
```

```
        (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))) (Suc ia))
         )
proof auto
 fix i
 fix ia
 assume a0: enat i < nlength ls
 assume a1: enat ia < nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
 show nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)))
          (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
          (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)) =
      nsubn σ (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia + nnth ls i)
           (nnth (nmap (λx. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
 proof −
  have 300: 0 < nlength σ
    by (metis assms(4) gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def)
  have 301: nnth ls i < nnth ls (Suc i)
   by (metis a0 assms(1) eSuc-enat ileI1 nidx-expand)
  have 302: (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f  fproj  g)
     by (simp add: a0 assms(5))
  have 303: the-enat (nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)))) = (nnth ls (Suc i)) − (nnth ls i)
      by (simp add: OPJ6help1 a0 assms(1) assms(2) assms(3) assms(4))
  have 304: nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) = (nnth ls (Suc i)) − (nnth ls i)
     using  a0 a1 cppl-fprojection[of f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))]
     using 302 303 by presburger
  have 305: (Suc ia) ≤ nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
    by (simp add: Suc-ile-eq a1)
  have 306: nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia ≤
           nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)
    by (metis 302 a1 cppl-fprojection eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)
  have 307: nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))=
           nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
               (the-enat(nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))))
     using 302 cppl-fprojection nnth-nlast by auto
  have 308: nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia) ≤
           nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
      by (metis 302 305 cppl-fprojection nidx-all-le-nlast)
  have 310: enat (nnth ls (Suc i)) ≤ nlength σ
    using a0 assms
     by (metis  enat-ord-simps(3) enat-the-enat nfinite-conv-nlength-enat  )
  have 30: nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia) ≤ nnth ls (Suc i) − nnth ls i
      using assms cppl-bounds[of  (nnth ls i) (nnth ls (Suc i)) σ f g Suc ia]
     PJ6help1[of ls σ ]  a0 a1 cppl-fprojection[of f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))]
      by (metis 303 308)
  have 311: nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)))
           (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
           (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)) =
          nsubn σ (nnth ls i + nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
                (nnth ls i + nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia))
    using nsubn-nsubn-1[of (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
      (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)) (nnth ls i) (nnth ls (Suc i)) σ ]
```

```
        using 30 301 306 310 order-less-imp-le by blast
    have 312: (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia + nnth ls i) =
            (nnth ls i + nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
        by simp
    have 313: (nnth ls i + nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) (Suc ia)) =
            (nnth (nmap (λx. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
      using 305 by force
    show ?thesis
    using 311 312 313 by presburger
  qed
 qed
have 4: ∀ i<nlength ls.
      ∀ ia<nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )).
      f (nsubn σ (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))) ia)
          (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))) (Suc ia))
           )
   using 2 3 by auto
have 5: nlength(lcppl f g σ ls) = nlength ls −1
    using assms  lcppl-nlength lcppl-nlength-zero
    by (simp add: epred-conv-minus lcppl-nlength-alt)
have 6: nlength σ >0 ⟶ nlength ls >0
    using assms
    by (simp add: nfinite-conv-nlength-enat)
have 7: ∀ i< nlength ls.
       ∀ ia<nlength ((nnth (lcppl f g σ ls) i)).
        f (nsubn σ (nnth (nnth (lcppl f g σ ls) i) ia)
             (nnth (nnth (lcppl f g σ ls) i) (Suc ia))
             )
    using assms 4 lcppl-nnth
    by (metis nlength-nmap)
have 8: (∀ i≤ nlength(lcppl f g σ ls).
         ∀ ia<nlength ((nnth (lcppl f g σ ls) i)).
          f (nsubn σ (nnth (nnth (lcppl f g σ ls) i) ia)
               (nnth (nnth (lcppl f g σ ls) i) (Suc ia))
               ))
 using 5 6 7 assms enat-ile nfinite-conv-nlength-enat not-le-imp-less by blast
have 9: powerinterval f σ (nfusecat ( (lcppl f g σ ls))) =
        (∀ i<nlength (nfusecat (lcppl f g σ ls)).
          f (nsubn σ (nnth (nfusecat (lcppl f g σ ls)) i) (nnth (nfusecat (lcppl f g σ ls)) (Suc i)) ))
 by (simp add: powerinterval-def)
have 10:  nlastnfirst (lcppl f g σ ls)
  using 6 assms lcppl-nfusecat-nlastnfirst-alt
  by (metis)
have 11: (∀  j ≤ nlength (lcppl f g σ ls). nlength(nnth (lcppl f g σ ls) j) > 0)
    using assms lcppl-nlength-all-gr-zero-alt by blast
have 110:  all-gr-zero (lcppl f g σ ls)
   using 11 all-gr-zero-nnth-a assms by blast
have 111:  all-nfinite (lcppl f g σ ls)
    using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt by blast
have 12: (∀ i≤ nlength(lcppl f g σ ls).
```

```
            (∀ ia<nlength ((nnth (lcppl f g σ ls) i)).
              f (nsubn σ (nnth (nnth (lcppl f g σ ls) i) ia)
                    (nnth (nnth (lcppl f g σ ls) i) (Suc ia))
                    ))) =
              (∀ j<nlength (nfusecat (lcppl f g σ ls)).
              f (nsubn σ (nnth (nfusecat (lcppl f g σ ls)) j)
                    (nnth (nfusecat (lcppl f g σ ls)) (Suc j))
                    ))

    using nfusecat-split-nsubn[of (lcppl f g σ ls) f σ] 10 11 110 111 assms
    unfolding nridx-expand
    by (simp add: Suc-ile-eq)
  show ?thesis using 12 8 9 using assms
  by meson
qed

lemma PJ7helpchain1a-help-3:
 assumes nidx ls
        nnth ls 0 = 0
        nfinite ls
        nfinite σ
         nlast ls = the-enat(nlength σ)
         (∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f fproj g)
         h (pfilt σ ls)
         nlength σ >0
 shows   nlast (nfusecat (lcppl f g σ ls)) = nlength σ
proof −
 have 0: nlength σ > 0 ⟶ nlength ls >0
   using assms
   by (metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero)
 have 00: nfinite (lcppl f g σ ls)
      using assms(1) assms(3) lcppl-nfinite by blast
 have 01: nlastnfirst (lcppl f g σ ls)
   using 0 assms lcppl-nfusecat-nlastnfirst by blast
 have 02: all-nfinite (lcppl f g σ ls)
   using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite by blast
 have 1: nlength σ >0 ⟶
           nlast (nfusecat (lcppl f g σ ls)) = nlast(nlast ( (lcppl f g σ ls)))
     using assms 0 nlastfirst-nfusecat-nlast[of (lcppl f g σ ls) ] 00 01 02 by blast
 have 2: nlength σ >0 ⟶
        nlast ( (lcppl f g σ ls))
         = (nnth (lcppl f g σ ls) (the-enat(nlength ls) −1))
      using assms
      by (metis 0 epred-enat lcppl-nfinite lcppl-nlength nfinite-nlength-enat nnth-nlast
          the-enat.simps)
 have 3: nlength σ >0 ⟶
         (nnth (lcppl f g σ ls) (the-enat(nlength ls) −1)) =
          (nmap (λx. x+ (nnth ls (the-enat(nlength ls) −1)))
              (cppl f g (nsubn σ (nnth ls (the-enat(nlength ls) −1))
                            (nnth ls (Suc (the-enat(nlength ls) −1))) )))
```

using *assms 0 lcppl-nnth*[*of ls* (*the-enat*(*nlength ls*) −*1*) *f g σ*]
by (*metis Suc-n-not-le-n add.commute diff-add enat.distinct*(*2*) *enat-ord-simps*(*1*)
enat-ord-simps(*4*) *enat-the-enat ileI1 not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc*)
have *4*: *nlength σ* >*0* ⟶
nlast(((*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))) =
the-enat(*nlength* ((*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))
using *0 assms cppl-fprojection*[*of f g*
(*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*)) (*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))]
by (*metis Suc-n-not-le-n add.commute diff-add enat.distinct*(*2*) *enat-ord-simps*(*1*)
enat-the-enat ileI1 nfinite-conv-nlength-enat not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc)

have *5*: *nlength σ* >*0* ⟶
the-enat(*nlength* ((*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*)))))) =
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) − (*nnth ls* (*the-enat*(*nlength ls*) −*1*))

using *0 PJ6help1 assms*
by (*metis Suc-n-not-le-n add.commute diff-add enat-ord-simps*(*1*) *ileI1 le-add-same-cancel1*
nfinite-conv-nlength-enat not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc the-enat.simps)
have *60*:*nfinite* (*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat* (*nlength ls*) − *1*))
(*nnth ls* (*Suc* (*the-enat* (*nlength ls*) − *1*)))))
by (*metis 0 Suc-n-not-le-n add.commute assms*(*3*) *assms*(*6*) *assms*(*8*) *cppl-fprojection diff-add*
enat.distinct(*2*) *enat-ord-simps*(*1*) *enat-the-enat ileI1 nfinite-conv-nlength-enat*
not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc)
have *6*: *nlength σ* >*0* ⟶
nlast ((*nmap* (λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
(*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))) =
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
(*nlast* (*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))
using *nlast-nmap*[*of* (*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*)))))
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))] *60* **by** *blast*
have *7*: *nlength σ* >*0* ⟶
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
(*nlast* (*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*)))))) =
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
((*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) − (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
using *4 5* **by** *auto*
have *8*: *nlength σ* >*0* ⟶
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
((*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) − (*nnth ls* (*the-enat*(*nlength ls*) −*1*))) =
((*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) −
(*nnth ls* (*the-enat*(*nlength ls*) −*1*))) + (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
by (*simp add*: *shift-def*)

have *9*: *nlength σ >0 ⟶*
((nnth ls (Suc (the-enat(nlength ls) −1))) −
(nnth ls (the-enat(nlength ls) −1))) + (nnth ls (the-enat(nlength ls) −1))
= (nnth ls (Suc (the-enat(nlength ls) −1)))
using *assms 0*
by (*metis Suc-diff-1 cancel-comm-monoid-add-class.diff-cancel diff-add diff-is-0-eq enat-ord-simps(1) enat-ord-simps(2) nfinite-nlength-enat nidx-expand order-less-imp-le the-enat.simps zero-enat-def*)
have *10*: *nlength σ >0 ⟶ (nnth ls (Suc (the-enat(nlength ls) −1))) = nlength σ*
using *assms 0*
by (*metis One-nat-def add.commute diff-add eSuc-enat enat.distinct(2) enat-ord-simps(1) enat-the-enat ileI1 nfinite-conv-nlength-enat nnth-nlast plus-1-eq-Suc zero-enat-def*)
show *?thesis*
using *1 10 2 3 6 7 8 9 assms* **by** *presburger*
qed

lemma *PJ7helpchain1a-help-4*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
h (pfilt σ ls)
nlength σ >0
shows *((pfilt σ (nfusecat (lcppl f g σ ls))) ⊨ g fproj h)*
proof −
have *0*: *nlength σ > 0 ⟶ nlength ls >0*
using *assms*
by (*metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *00*: *all-gr-zero (lcppl f g σ ls)*
using *all-gr-zero-nnth-a assms lcppl-nlength-all-gr-zero* **by** *blast*
have *01*: *all-nfinite (lcppl f g σ ls)*
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite* **by** *blast*
have *02*: *nnth (addzero (lsum (lcppl f g σ ls) 0)) 0 = 0*
using *0 assms*
by (*metis addzero-def lcppl-lsum-nlength less-numeral-extra(3) nnth-0*)
have *1*: *nlength σ >0 ⟶ nidx (addzero (lsum (lcppl f g σ ls) 0)) ∧ nnth (addzero (lsum (lcppl f g σ ls) 0)) 0 = 0*
using *assms lsum-addzero-nidx*[*of (lcppl f g σ ls)*] *lcppl-nlength-all-gr-zero*[*of ls σ f g*]
using *00 01 02* **by** *blast*
have *2*: *nlength σ > 0 ⟶*
nlast (addzero (lsum (lcppl f g σ ls) 0)) =
the-enat(nlength (pfilt σ (nfusecat (lcppl f g σ ls))))
using *assms lcppl-lsum-nlength*[*of ls σ f g*] *lcppl-nfusecat-pfilt-nlength*[*of ls σ f g*]
by *fastforce*
have *3*: *nlength σ >0 ⟶*
powerinterval g (pfilt σ (nfusecat (lcppl f g σ ls))) (addzero (lsum (lcppl f g σ ls) 0))
by (*simp add*: *assms lcppl-nfusecat-pfilt-powerinterval*)
have *4*: *nlength σ >0 ⟶*

$h\ (pfilt\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)))$
by (*simp add*: *assms pfilt-nmap-pfilt lcppl-pfilt-nfusecat-lsum*)
show *?thesis* **using** *1 2 3 4 assms fprojection-d-def*
by (*metis 0 00 01 lcppl-lsum-nlength lcppl-nfinite lcppl-nfusecat-nlastnfirst*
nfinite-conv-nlength-enat nfinite-nmap nfusecat-nfinite pfilt-nmap)
qed

lemma *PJ7helpchain1a-help-4-alt*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
$\neg nfinite\ ls$
$\neg nfinite\ \sigma$
$(\forall\ i{<}nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models f\ \ fproj\ \ g)$
$h\ (pfilt\ \sigma\ ls)$
shows $((pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))) \models g\ oproj\ h)$
proof −
have *00*: *all-gr-zero* $(lcppl\ f\ g\ \sigma\ ls)$
using *all-gr-zero-nnth-a assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *01*: *all-nfinite* $(lcppl\ f\ g\ \sigma\ ls)$
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt* **by** *blast*
have *02*: $nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ 0 = 0$
using *assms*
by (*metis addzero-def lcppl-nfinite nfinite-lsum-conv-b nlength-eq-enat-nfiniteD nnth-0 zero-enat-def*)
have *1*: $nidx\ \ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) \wedge nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ 0 = 0$
using *assms lsum-addzero-nidx 00 01 02* **by** *blast*
have *3*: $powerinterval\ g\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
by (*simp add*: *assms lcppl-nfusecat-pfilt-powerinterval-alt*)
have *4*: $h\ (pfilt\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)))$
by (*simp add*: *assms lcppl-pfilt-nfusecat-lsum-alt pfilt-nmap-pfilt*)
have *5*: $\neg\ nfinite\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
using *assms*
by (*simp add*: *lcppl-lsum-nlength-alt nfinite-conv-nlength-enat*)
have *6*: $\neg nfinite\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))$
using *assms 00 01 lcppl-nfinite lcppl-nfusecat-nlastnfirst-alt nfusecat-nfinite* **by** *blast*
have *7*: $\neg\ nfinite\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
by (*simp add*: *6 pfilt-nmap*)
show *?thesis* **using** *1 3 4 assms* **unfolding** *oprojection-d-def*
using *5 7* **by** *blast*
qed

lemma *PJ7helpchain1a*:
assumes $nlength\ \sigma > 0$
$(\sigma \models (f\ \ fproj\ \ g)\ \ fproj\ \ h)$
shows $(\sigma \models f\ \ fproj\ \ (g\ \ fproj\ \ h))$
proof −
have *1*: $nlength\ \sigma > 0$
using *assms* **by** *auto*
have *2*: $(\exists ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$powerinterval\ (LIFT(f\ \ fproj\ \ g))\ \sigma\ ls \wedge$
$h\ (pfilt\ \sigma\ ls))$

using *assms* **using** *cppl-fprojection*[*of LIFT*(*f fproj g*) *h σ*] **by** *blast*
obtain *ls* **where** *3*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite σ* ∧
nlast ls = *the-enat*(*nlength σ*) ∧
powerinterval (*LIFT*(*f fproj g*)) *σ ls* ∧
h (*pfilt σ ls*)
using *2* **by** *blast*
have *4*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite σ* ∧ *nlast ls* = *the-enat*(*nlength σ*) ∧
(∀ *i* <*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*) ∧
h (*pfilt σ ls*)
using *3* **by** (*simp add*: *powerinterval-def*)
have *6*: *nlength ls>0*
using *1 4*
by (*metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *7*: *nidx* (*nfusecat* (*lcppl f g σ ls*)) ∧ *nnth* (*nfusecat* (*lcppl f g σ ls*)) *0* = *0*
by (*metis* (*no-types*, *lifting*) *1 4 PJ7helpchain1a-help-1*)
have *8*: *powerinterval f σ* (*nfusecat* (*lcppl f g σ ls*))
using *4 6 PJ7helpchain1a-help-2 assms* **by** *auto*
have *9*: *nlast* (*nfusecat* (*lcppl f g σ ls*)) = *the-enat*(*nlength σ*)
by (*metis 1 4 PJ7helpchain1a-help-3 the-enat.simps*)
have *10*: ((*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) ⊨ *g fproj h*)
using *1 4 PJ7helpchain1a-help-4* **by** *blast*
show *?thesis*
using *10 7 8 9* **by** (*metis 3 fprojection-d-def nfinite-nmap pfilt-nmap*)
qed

lemma *OPJ7helpchain1a*:
assumes (*σ* ⊨ (*f fproj g*) *oproj h*)
shows (*σ* ⊨ *f oproj* (*g oproj h*))
proof −
have *2*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite σ* ∧
powerinterval (*LIFT*(*f fproj g*)) *σ ls* ∧
h (*pfilt σ ls*))
using *assms* **using** *cppl-oprojection*[*of LIFT*(*f fproj g*) *h σ*] **by** *blast*
obtain *ls* **where** *3*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite σ* ∧
powerinterval (*LIFT*(*f fproj g*)) *σ ls* ∧
h (*pfilt σ ls*)
using *2* **by** *blast*
have *4*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite σ* ∧
(∀ *i* <*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*) ∧
h (*pfilt σ ls*)
using *3* **by** (*simp add*: *powerinterval-def*)
have *6*: *nlength ls>0*
by (*metis 4 gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def*)
have *7*: *nidx* (*nfusecat* (*lcppl f g σ ls*)) ∧ *nnth* (*nfusecat* (*lcppl f g σ ls*)) *0* = *0*
using *4 PJ7helpchain1a-help-1-alt*
by (*metis 6 OPJ6help4*)
have *8*: *powerinterval f σ* (*nfusecat* (*lcppl f g σ ls*))
using *4 PJ7helpchain1a-help-2-alt* **by** *blast*
have *10*: ((*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) ⊨ *g oproj h*)

using *4 PJ7helpchain1a-help-4-alt* **by** *blast*
show *?thesis*
using *10 7 8* **by** (*metis 3 nfinite-conv-nlength-enat oprojection-d-def pfilt-nlength*)
qed

lemma *PJ7helpchain1b*:
assumes *nlength* $\sigma > 0$
$(\sigma \models f\ fproj\ (g\ fproj\ h))$
shows $(\sigma \models (f\ fproj\ g)\ fproj\ h)$
proof −
have *1*: *nlength* $\sigma > 0$
using *assms* **by** *auto*
have *2*: $(\exists ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
powerinterval f σ *ls* $\wedge$
$((pfilt\ \sigma\ ls) \models g\ \ fproj\ \ h)\ )$
using *assms cppl-fprojection* **by** *blast*
obtain *ls* **where** *3*: $nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge\ \ nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
powerinterval f σ *ls* $\wedge$
$((pfilt\ \sigma\ ls) \models g\ \ fproj\ \ h)$
using *2* **by** *blast*
have *4*: $(\exists\ lsa.\ nidx\ \ lsa \wedge\ \ nnth\ lsa\ 0 = 0 \wedge nfinite\ lsa \wedge nfinite\ (pfilt\ \sigma\ ls) \wedge$
$nlast\ \ lsa\ = the\text{-}enat(nlength(pfilt\ \sigma\ ls)) \wedge$
powerinterval g $(pfilt\ \sigma\ ls)$ *lsa* $\wedge$
$((pfilt\ (pfilt\ \sigma\ ls)\ lsa) \models h))$
using *3* **using** *cppl-fprojection* **by** *blast*
obtain *lsa* **where** *5*: $nidx\ \ lsa \wedge nnth\ lsa\ 0 = 0 \wedge nfinite\ lsa \wedge nfinite\ (pfilt\ \sigma\ ls) \wedge$
$nlast\ lsa = the\text{-}enat(nlength(pfilt\ \sigma\ ls)) \wedge$
powerinterval g $(pfilt\ \sigma\ ls)$ *lsa* $\wedge$
$((pfilt\ (pfilt\ \sigma\ ls)\ lsa) \models h)$
using *4* **by** *blast*
have *6*: *nlength ls* >0
using *1 3*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *7*: $nlength(pfilt\ \sigma\ ls) = nlength\ ls$
by (*simp add*: *pfilt-nlength*)
have *8*: $(pfilt\ (pfilt\ \sigma\ ls)\ lsa) = (pfilt\ \sigma\ (pfilt\ ls\ lsa))$
using *pfilt-nmap-pfilt* **by** *blast*
have *9*: $nlast\ (pfilt\ ls\ lsa)\ = the\text{-}enat(nlength\ \sigma)$
by (*metis 3 5 7 nlast-nmap nnth-nlast pfilt-nmap*)
have *10*: *nlength lsa* >0
using *5 6 7*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *11*: $nlength\ (pfilt\ ls\ lsa) > 0$
by (*metis 10 pfilt-nlength*)
have *111*: $nnth\ (pfilt\ ls\ lsa)\ 0 = 0$
by (*metis 3 5 pfilt-nnth zero-enat-def zero-le*)
have *112*: $nlength(pfilt\ ls\ lsa) = nlength\ lsa$
using *pfilt-expand* **by** *blast*

have *113*: $(\forall\ i{<}\ nlength\ lsa.\ (nnth\ (pfilt\ ls\ lsa)\ i) = (nnth\ ls\ (nnth\ lsa\ i)))$
by (*simp add*: *pfilt-nmap*)
have *114*: $(\forall\ i{<}\ nlength\ lsa.\ (nnth\ (pfilt\ ls\ lsa)\ (Suc\ i)) = (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
by (*metis 112 eSuc-enat ileI1 pfilt-nnth*)
have *1141*: $\bigwedge j.\ j \leq nlength\ lsa \longrightarrow nnth\ lsa\ j \leq nlength\ ls$
by (*metis 5 7 enat-ord-simps*(*1*) *ndropn-eq-NNil ndropn-nlast nfinite-conv-nlength-enat nidx-less-eq the-enat.simps*)
have *115*: $(\forall\ i{<}\ nlength\ lsa.\ (nnth\ ls\ (nnth\ lsa\ i)) < (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))\ )$
by (*metis 1141 3 5 eSuc-enat ileI1 less-imp-Suc-add nidx-expand nidx-less*)
have *12*: $nidx\ \ (pfilt\ ls\ lsa) \wedge nnth\ (pfilt\ ls\ lsa)\ 0 = 0$
by (*simp add*: *111 112 115 Suc-ile-eq nidx-expand pfilt-nnth*)
have *2021*: $(\forall\ i{<}\ nlength\ lsa.\ nfinite\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))))$
by (*simp add*: *nsubn-def1*)
have *2022*: $\bigwedge i.\ i{<}\ nlength\ lsa \Longrightarrow (nnth\ ls\ (nnth\ lsa\ i)) \leq nlength\ \sigma$
by (*metis 3 5 7 enat.distinct*(*2*) *enat-ord-simps*(*1*) *enat-the-enat nfinite-conv-nlength-enat nidx-all-le-nlast order-less-imp-le*)
have *2023*: $\bigwedge j.\ j \leq nlength\ lsa \longrightarrow nnth\ lsa\ j \leq nlength\ ls$
by (*metis 5 7 enat-ord-simps*(*1*) *ndropn-eq-NNil ndropn-nlast nfinite-conv-nlength-enat nidx-less-eq the-enat.simps*)
have *203*: $(\forall\ i{<}\ nlength\ lsa.\ \ nlength\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))\ ) = (nnth\ ls\ (nnth\ lsa\ (Suc\ i))) - (nnth\ ls\ (nnth\ lsa\ i)))$
by (*metis 112 113 114 12 3 5 9 PJ6help1 le-add1 le-add-same-cancel1 nfinite-conv-nlength-enat*)
have *2041*: $(\forall\ i{<}\ nlength\ lsa.\ \ (nnth\ lsa\ (Suc\ i)) \leq nlength\ ls\ )$
by (*metis 2023 eSuc-enat ileI1*)
have *204*: $(\forall\ i{<}\ nlength\ lsa.\ nlength\ (nmap\ (\lambda x.\ x{-}\ (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )) = (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i))$
by (*simp add*: *3 5 PJ6help1 pfilt-nlength*)
have *205*: $(\forall\ i{<}\ nlength\ lsa.\ (\forall\ j \leq (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i).\ (nnth\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )\ j) = (nnth\ ls\ ((nnth\ lsa\ i) + j))\ )\ )$
using *2041 5* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *2060*: $(\forall\ i{<}\ nlength\ lsa.\ (nnth\ lsa\ i) \leq (nnth\ lsa\ (Suc\ i))\ )$
using *5* **by** (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *206*: $(\forall\ i{<}\ nlength\ lsa.\ (\forall\ j \leq (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i).\ (nnth\ ls\ (nnth\ lsa\ i)) \leq (nnth\ ls\ ((nnth\ lsa\ i) + j))\ )\ )$
using *2041 3 2060*
by (*metis le-add1 nidx-all-le-nlast nidx-less-eq nnth-beyond not-le-imp-less order-less-imp-le*)
have *207*: $(\forall\ i{<}\ nlength\ lsa.\ (\forall\ j \leq (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i).\ (nnth\ (nmap\ (\lambda x.\ x{-}\ (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ ))\ j\ ) = (nnth\ ls\ ((nnth\ lsa\ i) + j)) - (nnth\ ls\ (nnth\ lsa\ i)))$

```
                ))
  using 204 205 by auto
 have 208: (∀ i< nlength lsa.
                (nnth (nmap (λx. x− (nnth ls (nnth lsa i)))
                          (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) 0) = 0 )
  by (simp add: 207)
 have 209: (∀ i< nlength lsa.
                nlast (nmap (λx. x− (nnth ls (nnth lsa i)))
                          (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) =
                (nnth ls (nnth lsa (Suc i))) − (nnth ls (nnth lsa i))
              )

  using 204 207 5 2060 by (simp add: nsubn-def1 ntaken-ndropn-nlast)
 have 210: (∀ i< nlength lsa.
            nidx (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) ∧
            nnth (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) 0 = 0
             )
  using 3 5 7 nidx-expand nidx-shiftm nidx-nsubn by (simp add: 2041 Suc-ile-eq order-less-imp-le)
 have 20: powerinterval (LIFT(f fproj g)) σ (pfilt ls lsa)
   proof −
    have 201: powerinterval (LIFT(f fproj g)) σ (pfilt ls lsa) =
            (∀ i< nlength lsa. (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ⊨ f fproj g)
          unfolding powerinterval-def by (auto simp add: Suc-ile-eq pfilt-nlength pfilt-nnth)
    have 202: (∀ i< nlength lsa. (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ⊨ f fproj
g) =
              (∀ i< nlength lsa.
                (∃ ls1. nidx ls1 ∧ nnth ls1 0 = 0 ∧ nfinite ls1 ∧
                 nfinite (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ∧
                 nlast ls1 = the-enat(nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) )) ∧
                  powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1 ∧
                  ((pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1) ⊨ g)
                  ))

     by (simp add: fprojection-d-def)
    have fg: (∀ i< nlength lsa.
                (∃ ls1. nidx ls1 ∧ nnth ls1 0 = 0 ∧ nfinite ls1 ∧
                 nfinite (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ∧
                 nlast ls1 = the-enat(nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) )) ∧
                  powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1 ∧
                  ((pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1) ⊨ g)
                  ))
    proof auto
     fix i::nat
     assume a: enat i < nlength lsa
     show ∃ls1. nidx ls1 ∧
            nnth ls1 0 = 0 ∧
            nfinite ls1 ∧
            nfinite (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ∧
            nlast ls1 = the-enat (nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))) ∧
            powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ls1 ∧
```

```
      g (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ls1)
    proof −
   have fg1: nnth (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) 0 = 0
     using 210 a by blast
    have fg2: nidx (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))
     using 210 a by blast
    have fg3: nfinite (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
     using 2021 a by blast
    have fg4: nlast (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) =
           the-enat (nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))
     using 203 209 a the-enat.simps by presburger
    have fg5: powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
            (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))
    proof (auto simp add: powerinterval-def)
     fix ia::nat
     assume aa: enat ia < nlength (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))
     show f (nsubn (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
              (nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) ia − nnth ls (nnth lsa i))
               (nnth (nmap (λx. x − nnth ls (nnth lsa i)) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))))
(Suc ia)))
      proof −
       have f1: ia < (nnth lsa (Suc i)) − (nnth lsa i)
         by (metis 5 7 PJ6help1 a aa enat-ord-simps(2) le-add1 le-add-same-cancel1 nsubn-nlength)
       have f2: nsubn (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                 (nnth ls (nnth lsa i + ia) − nnth ls (nnth lsa i))
                 (nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i)) =
              nsubn σ (nnth ls (nnth lsa i + ia)) (nnth ls (nnth lsa i + Suc ia))
       proof −
        have f3: (nnth ls (nnth lsa i + ia) − nnth ls (nnth lsa i) ≤
                    nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i))
          by (metis 2041 205 2060 3 Suc-leI a aa diff-le-mono eSuc-enat f1 ileI1
                le-add2 nidx-less-eq nidx-nsubn order-less-imp-le plus-1-eq-Suc)
        have f4: nnth ls (nnth lsa i) ≤ nnth ls (nnth lsa (Suc i))
          using 115 a dual-order.order-iff-strict by blast
        have f5: enat (nnth ls (nnth lsa (Suc i))) ≤ nlength σ
          by (metis 2041 3 a enat-ord-code(4) enat-ord-simps(1) enat-the-enat nidx-all-le-nlast
              order-less-imp-le)
        have f6: nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i) ≤
                    nnth ls (nnth lsa (Suc i)) − nnth ls (nnth lsa i)
          by (metis 2041 2060 3 Suc-leI a add-diff-cancel-left' diff-le-mono f1 le-add1
              le-diff-iff nidx-less-eq)
        show ?thesis using nsubn-nsubn-1[of (nnth ls (nnth lsa i + ia) − nnth ls (nnth lsa i))
                                            (nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i))
                (nnth ls (nnth lsa i))  (nnth ls (nnth lsa (Suc i))) σ ]
          by (metis 206 Suc-leI a f1 f3 f4 f5 f6 le-add-diff-inverse order-less-imp-le)
       qed
       have f7: (nnth (nmap (λx. x − nnth ls (nnth lsa i))
                              (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))) (Suc ia)) =
                 (nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) (Suc ia) − nnth ls (nnth lsa i))
        by (metis aa eSuc-enat ileI1 nnth-nmap)
```

```
 have f8: f (nsubn σ (nnth ls ((nnth lsa i) +ia)) ((nnth ls ((nnth lsa i)+(Suc ia)))) )
  using 2041 3 unfolding powerinterval-def
    by (metis a add.commute add-Suc-right enat-ord-simps(2) f1 less-diff-conv
        order-less-le-trans )
 show ?thesis using 205 a f1 f2 f7 f8 by fastforce
qed
qed
have fg6: g (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
             (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )))
 proof −
  have g1: (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
             (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))) =
               nmap (nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))
                    (nmap (λx. x − nnth ls (nnth lsa i)) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))))
    using pfilt-nmap[of (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))]
    by blast
  have g2: ... =
             nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))))) ∘
                  (λx. x− (nnth ls (nnth lsa i))))
                 (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))
    using nellist.map-comp by blast
  have g3: ... =
             (nsubn (nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))))) ∘
                    (λx. x− (nnth ls (nnth lsa i)))) ls)
                 (nnth lsa i) (nnth lsa (Suc i)))
    unfolding nsubn-def1 by (simp add: ndropn-nmap)
  have g4: ⋀y. (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))))) ∘
                    (λx. x− (nnth ls (nnth lsa i)))) y =
                nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                     (y − nnth ls (nnth lsa i))
     by simp
  have g5: (nsubn (nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))))) ∘
                    (λx. x− (nnth ls (nnth lsa i)))) ls)
                 (nnth lsa i) (nnth lsa (Suc i))) =
             (nsubn (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                                    (y − nnth ls (nnth lsa i))) ls)
                 (nnth lsa i) (nnth lsa (Suc i)))
    using g4 by presburger
  have g6: ⋀j. (nnth lsa i) ≤j ∧ j ≤ (nnth lsa (Suc i)) ⟶
               nnth (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                                    (y − nnth ls (nnth lsa i))) ls) j =
               nnth (nmap (nnth σ) ls) j
    proof
     fix j::nat
     assume aaa: nnth lsa i ≤ j ∧ j ≤ nnth lsa (Suc i)
     show nnth (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                              (y − nnth ls (nnth lsa i))) ls) j = nnth (nmap (nnth σ) ls) j
     proof −
      have g8: nnth (nmap (nnth σ) ls) j = nnth σ (nnth ls j)
```

```
      by (metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
    have g9: nnth (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                        (y − nnth ls (nnth lsa i))) ls) j =
              (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                        (y − nnth ls (nnth lsa i))) (nnth ls j)
      by (metis (no-types, lifting) nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
    have g10: ... =
              nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                (nnth ls j − nnth ls (nnth lsa i))
      by auto
    have g11: ... = nnth σ ((nnth ls (nnth lsa i)) + (nnth ls j − nnth ls (nnth lsa i)))
      using nsubn-nnth[of σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))
        (nnth ls j − nnth ls (nnth lsa i))]
      by (simp add: 2041 3 a aaa diff-le-mono nidx-less-eq)
    have g12: ... = nnth σ (nnth ls j)
      using aaa
      by (metis 3 le-add-diff-inverse nidx-all-le-nlast nidx-less-eq nnth-beyond
          not-le-imp-less order-less-imp-le)
    show ?thesis
    using g11 g12 g8 g9 by presburger
   qed
  qed
 have g13: (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
            (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))) =
             (nsubn (pfilt σ ls) (nnth lsa i) (nnth lsa (Suc i)) )
             by (simp add: 2041 2060 a g2 g3 g5 g6 nsubn-eq pfilt-nmap)
 show ?thesis using 5 a by (simp add: g13 powerinterval-def)
 qed
show ?thesis
using 203 2041 2060 209 210 a fg3 fg5 fg6 nsubn-nfinite by fastforce
qed
qed
show ?thesis
using 201 202 fg by blast
qed
show ?thesis
by (metis 12 20 3 5 8 9 fprojection-d-def nfinite-nmap pfilt-nmap)
qed


lemma OPJ7helpchain1b:
assumes (σ ⊨ f oproj (g oproj h))
shows   (σ ⊨ (f fproj g) oproj h)
proof −
 have 2: (∃ ls. nidx  ls ∧ nnth ls 0 = 0 ∧ ¬nfinite ls ∧ ¬nfinite σ ∧
        powerinterval f σ ls ∧
        ((pfilt σ ls) ⊨ g  oproj  h) )
  using assms cppl-oprojection by blast
 obtain ls where 3: nidx  ls ∧ nnth ls 0 = 0 ∧ ¬nfinite ls ∧ ¬nfinite σ ∧
        powerinterval f σ ls ∧
```

$((pfilt\ \sigma\ ls) \models g\ \ oproj\ \ h)$
using *2* **by** *blast*
have *4*: $(\exists\ lsa.\ nidx\ \ lsa \wedge\ \ nnth\ lsa\ 0 = 0 \wedge \neg nfinite\ lsa \wedge \neg nfinite\ (pfilt\ \sigma\ ls) \wedge$
$powerinterval\ g\ (pfilt\ \sigma\ ls)\ lsa \wedge$
$((pfilt\ (pfilt\ \sigma\ ls)\ lsa) \models h))$
using *3* **using** *cppl-oprojection* **by** *blast*
obtain *lsa* **where** *5*: $nidx\ \ lsa \wedge nnth\ lsa\ 0 = 0 \wedge \neg nfinite\ lsa \wedge \neg nfinite\ (pfilt\ \sigma\ ls) \wedge$
$powerinterval\ g\ (pfilt\ \sigma\ ls)\ lsa \wedge$
$((pfilt\ (pfilt\ \sigma\ ls)\ lsa) \models h)$
using *4* **by** *blast*
have *7*: $nlength(pfilt\ \sigma\ ls) = nlength\ ls$
by (*simp add*: *pfilt-nlength*)
have *8*: $(pfilt\ (pfilt\ \sigma\ ls)\ lsa) = (pfilt\ \sigma\ (pfilt\ ls\ lsa))$
using *pfilt-nmap-pfilt* **by** *blast*
have *11*: $nlength\ (pfilt\ ls\ lsa) > 0$
by (*metis 5 gr-zeroI nlength-eq-enat-nfiniteD pfilt-nlength zero-enat-def*)
have *111*: $nnth\ (pfilt\ ls\ lsa)\ 0 = 0$
by (*metis 3 5 pfilt-nnth zero-enat-def zero-le*)
have *112*: $nlength(pfilt\ ls\ lsa) = nlength\ lsa$
using *pfilt-expand* **by** *blast*
have *113*: $(\forall\ i{<}\ nlength\ lsa.\ (nnth\ (pfilt\ ls\ lsa)\ i) = (nnth\ ls\ (nnth\ lsa\ i)))$
by (*simp add*: *pfilt-nmap*)
have *114*: $(\forall\ i{<}\ nlength\ lsa.\ (nnth\ (pfilt\ ls\ lsa)\ (Suc\ i)) = (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
by (*metis 112 eSuc-enat ileI1 pfilt-nnth*)
have *1141*: $\bigwedge j.\ j \leq nlength\ lsa \longrightarrow nnth\ lsa\ j \leq nlength\ ls$
by (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
have *115*: $(\forall\ i{<}\ nlength\ lsa.\ (nnth\ ls\ (nnth\ lsa\ i)) < (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))\ )$
by (*metis 1141 3 5 eSuc-enat ileI1 less-imp-Suc-add nidx-expand nidx-less*)
have *12*: $nidx\ \ (pfilt\ ls\ lsa) \wedge nnth\ (pfilt\ ls\ lsa)\ 0 = 0$
by (*simp add*: *111 112 115 Suc-ile-eq nidx-expand pfilt-nnth*)
have *2021*: $(\forall\ i{<}\ nlength\ lsa.\ nfinite\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))))$
by (*simp add*: *nsubn-def1*)
have *2022*: $\bigwedge i.\ i{<}\ nlength\ lsa \Longrightarrow (nnth\ ls\ (nnth\ lsa\ i)) \leq nlength\ \sigma$
by (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
have *203*: $(\forall\ i{<}\ nlength\ lsa.\ \ nlength\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))\ ) =$
$(nnth\ ls\ (nnth\ lsa\ (Suc\ i))) - (nnth\ ls\ (nnth\ lsa\ i)))$
by (*metis 112 113 114 12 3 5 OPJ6help1 le-add1 le-add-same-cancel1 nfinite-nmap*
pfilt-nmap)
have *2041*: $(\forall\ i{<}\ nlength\ lsa.\ \ (nnth\ lsa\ (Suc\ i)) \leq nlength\ ls\ )$
by (*metis 1141 eSuc-enat ileI1*)
have *204*: $(\forall\ i{<}\ nlength\ lsa.$
$nlength\ (nmap\ (\lambda x.\ x{-}\ (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )) =$
$(nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i))$
by (*simp add*: *3 5 OPJ6help1*)
have *205*: $(\forall\ i{<}\ nlength\ lsa.$
$(\forall\ j \leq (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i).$
$(nnth\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )\ j) =$
$(nnth\ ls\ ((nnth\ lsa\ i) + j))$
$)$
$)$

using *2041 5* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *2060*: (∀ *i*< *nlength lsa*. (*nnth lsa i*) ≤ (*nnth lsa* (*Suc i*)))
using *5* **by** (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *206*: (∀ *i*< *nlength lsa*.
(∀ *j* ≤ (*nnth lsa* (*Suc i*)) − (*nnth lsa i*).
(*nnth ls* (*nnth lsa i*)) ≤ (*nnth ls* ((*nnth lsa i*) +*j*))
)
)
using *2041 3 2060* **by** (*simp add*: *nfinite-conv-nlength-enat nidx-less-eq*)
have *207*: (∀ *i*< *nlength lsa*.
(∀ *j* ≤ (*nnth lsa* (*Suc i*)) − (*nnth lsa i*).
(*nnth* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *j*) =
(*nnth ls* ((*nnth lsa i*) +*j*)) − (*nnth ls* (*nnth lsa i*))
))
using *204 205* **by** *auto*
have *208*: (∀ *i*< *nlength lsa*.
(*nnth* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0*) = *0*)
by (*simp add*: *207*)
have *209*: (∀ *i*< *nlength lsa*.
nlast (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) =
(*nnth ls* (*nnth lsa* (*Suc i*))) − (*nnth ls* (*nnth lsa i*))
)
using *204 207 5 2060* **by** (*simp add*: *nsubn-def1 ntaken-ndropn-nlast*)
have *210*: (∀ *i*< *nlength lsa*.
nidx (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) ∧
nnth (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0* = *0*
)
using *3 5 7 nidx-expand nidx-shiftm nidx-nsubn* **by** (*simp add*: *2041 Suc-ile-eq order-less-imp-le*)
have *20*: *powerinterval* (*LIFT*(*f fproj g*)) *σ* (*pfilt ls lsa*)
proof −
have *201*: *powerinterval* (*LIFT*(*f fproj g*)) *σ* (*pfilt ls lsa*) =
(∀ *i*< *nlength lsa*. (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ⊨ *f fproj g*)
unfolding *powerinterval-def* **by** (*auto simp add*: *Suc-ile-eq pfilt-nlength pfilt-nnth*)
have *202*: (∀ *i*< *nlength lsa*. (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ⊨ *f fproj*
g) =
(∀ *i*< *nlength lsa*.
(∃ *ls1*. *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧
nfinite (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ∧
nlast ls1 = *the-enat*(*nlength* (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) ∧
powerinterval f (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1* ∧
((*pfilt* (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1*) ⊨ *g*)
))
by (*simp add*: *fprojection-d-def*)
have *fg*: (∀ *i*< *nlength lsa*.
(∃ *ls1*. *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧
nfinite (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ∧
nlast ls1 = *the-enat*(*nlength* (*nsubn σ* (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) ∧

```
                powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1 ∧
                ((pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1) ⊨ g)
                ))
    proof auto
     fix i::nat
     assume a: enat i < nlength lsa
     show ∃ls1. nidx ls1 ∧
           nnth ls1 0 = 0 ∧
           nfinite ls1 ∧
           nfinite (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ∧
           nlast ls1 = the-enat (nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))) ∧
           powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ls1 ∧
           g (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ls1)
      proof −
     have fg1: nnth (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) 0 = 0
         using 210 a by blast
      have fg2: nidx (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))
         using 210 a by blast
      have fg3: nfinite (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
        using 2021 a by blast
      have fg4: nlast (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) =
                the-enat (nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))
        using 203 209 a the-enat.simps by presburger
      have fg5: powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                   (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))
      proof (auto simp add: powerinterval-def)
       fix ia::nat
       assume aa: enat ia < nlength (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))
       show f (nsubn (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                    (nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) ia − nnth ls (nnth lsa i))
                     (nnth (nmap (λx. x − nnth ls (nnth lsa i)) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))))
(Suc ia)))
        proof −
         have f1: ia < (nnth lsa (Suc i)) − (nnth lsa i)
           by (metis 3 5 OPJ6help1 a aa enat-ord-simps(2) le-add1 le-add-same-cancel1)
         have f2: nsubn (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                     (nnth ls (nnth lsa i + ia) − nnth ls (nnth lsa i))
                     (nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i)) =
                nsubn σ (nnth ls (nnth lsa i + ia)) (nnth ls (nnth lsa i + Suc ia))
         proof −
          have f3: (nnth ls (nnth lsa i + ia) − nnth ls (nnth lsa i) ≤
                       nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i))
            by (metis 2041 205 2060 3 Suc-leI a aa diff-le-mono eSuc-enat f1 ileI1
                  le-add2 nidx-less-eq nidx-nsubn order-less-imp-le plus-1-eq-Suc)
          have f4: nnth ls (nnth lsa i) ≤ nnth ls (nnth lsa (Suc i))
            using 115 a dual-order.order-iff-strict by blast
          have f5: enat (nnth ls (nnth lsa (Suc i))) ≤ nlength σ
            by (metis 2022 5 a eSuc-enat ileI1 nfinite-conv-nlength-enat order-le-imp-less-or-eq)
          have f6: nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i) ≤
                       nnth ls (nnth lsa (Suc i)) − nnth ls (nnth lsa i)
```

```
    by (metis 2041 2060 3 Suc-leI a add-diff-cancel-left' diff-le-mono f1 le-add1
        le-diff-iff nidx-less-eq)
   show ?thesis using nsubn-nsubn-1[of (nnth ls (nnth lsa i + ia) − nnth ls (nnth lsa i))
                                       (nnth ls (nnth lsa i + Suc ia) − nnth ls (nnth lsa i))
          (nnth ls (nnth lsa i))  (nnth ls (nnth lsa (Suc i))) σ ]
    by (metis 206 Suc-leI a f1 f3 f4 f5 f6 le-add-diff-inverse order-less-imp-le)
  qed
  have f7: (nnth (nmap (λx. x − nnth ls (nnth lsa i))
                       (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))) (Suc ia)) =
            (nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) (Suc ia) − nnth ls (nnth lsa i))
    by (metis aa eSuc-enat ileI1 nnth-nmap)
  have f8: f (nsubn σ (nnth ls ((nnth lsa i) +ia)) ((nnth ls ((nnth lsa i)+(Suc ia)))) )
   using 2041 3 unfolding powerinterval-def
    by (metis a add.commute add-Suc-right enat-ord-simps(2) f1 less-diff-conv
        order-less-le-trans )
  show ?thesis using 205 a f1 f2 f7 f8 by fastforce
 qed
qed
have fg6: g (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
             (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )))
 proof −
  have g1: (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
             (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))) =
              nmap (nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))
                   (nmap (λx. x − nnth ls (nnth lsa i)) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))))
    using pfilt-nmap[of (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
       (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))]
    by blast
  have g2: ... =
             nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
                  (λx. x− (nnth ls (nnth lsa i))))
                 (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))
    using nellist.map-comp by blast
  have g3: ... =
             (nsubn (nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
                    (λx. x− (nnth ls (nnth lsa i)))) ls)
                  (nnth lsa i) (nnth lsa (Suc i)))
    unfolding nsubn-def1 by (simp add: ndropn-nmap)
  have g4: ⋀y. (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
                    (λx. x− (nnth ls (nnth lsa i)))) y =
                nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                     (y − nnth ls (nnth lsa i))
     by simp
  have g5: (nsubn (nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
                    (λx. x− (nnth ls (nnth lsa i)))) ls)
                  (nnth lsa i) (nnth lsa (Suc i))) =
             (nsubn (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                               (y − nnth ls (nnth lsa i))) ls)
                  (nnth lsa i) (nnth lsa (Suc i)))
    using g4 by presburger
```

```
      have g6: ⋀j. (nnth lsa i) ≤j ∧ j ≤ (nnth lsa (Suc i)) ⟶
                  nnth (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                                 (y − nnth ls (nnth lsa i))) ls) j =
                  nnth (nmap (nnth σ) ls) j
        proof
         fix j::nat
         assume aaa: nnth lsa i ≤ j ∧ j ≤ nnth lsa (Suc i)
         show nnth (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                        (y − nnth ls (nnth lsa i))) ls) j = nnth (nmap (nnth σ) ls) j
         proof −
          have g8: nnth (nmap (nnth σ) ls) j = nnth σ (nnth ls j)
            by (metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
          have g9: nnth (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                                 (y − nnth ls (nnth lsa i))) ls) j =
                        (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                                 (y − nnth ls (nnth lsa i))) (nnth ls j)
            by (metis (no-types, lifting) nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
          have g10: ... =
                      nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                        (nnth ls j − nnth ls (nnth lsa i))
            by auto
          have g11: ... = nnth σ ((nnth ls (nnth lsa i)) + (nnth ls j − nnth ls (nnth lsa i)))
            using nsubn-nnth[of σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))
              (nnth ls j − nnth ls (nnth lsa i)) ]
            by (simp add: 2041 3 a aaa diff-le-mono nidx-less-eq)
          have g12: ... = nnth σ (nnth ls j)
            using aaa
            by (metis 206 a add-le-imp-le-diff diff-add le-add-diff-inverse)
          show ?thesis
          using g11 g12 g8 g9 by presburger
         qed
        qed
       have g13: (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
                  (nmap (λx. x− (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))) =
                   (nsubn (pfilt σ ls) (nnth lsa i) (nnth lsa (Suc i)) )
                   by (simp add: 2041 2060 a g2 g3 g5 g6 nsubn-eq pfilt-nmap)
       show ?thesis using 5 a by (simp add: g13 powerinterval-def)
      qed
     show ?thesis
     using 203 2041 2060 209 210 a fg3 fg5 fg6 nsubn-nfinite by fastforce
    qed
   qed
  show ?thesis
  using 201 202 fg by blast
 qed
 show ?thesis
 by (metis 12 20 3 5 8 nfinite-nmap oprojection-d-def pfilt-nmap)
qed

lemma PJ7sem:
```

$(\sigma \models f \ \ fproj \ \ (g \ \ fproj \ \ h) = (f \ \ fproj \ \ g) \ \ fproj \ \ h)$
proof −
have *1*: $nlength\ \sigma > 0 \longrightarrow (\sigma \models f \ \ fproj \ \ (g \ \ fproj \ \ h) = (f \ \ fproj \ \ g) \ \ fproj \ \ h)$
using *PJ7helpchain1a PJ7helpchain1b unl-lift2* **by** *blast*
have *2*: $nlength\ \sigma = 0 \longrightarrow (\sigma \models f \ \ fproj \ \ (g \ \ fproj \ \ h) = (f \ \ fproj \ \ g) \ \ fproj \ \ h)$
using *PJ7empty* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *OPJ7sem*:
$(\sigma \models f\ oproj\ (g\ oproj\ h) = (f \ \ fproj \ \ g)\ oproj\ h)$
using *OPJ7helpchain1a OPJ7helpchain1b unl-lift2* **by** *blast*

PJ8

lemma *PJ8semhelp*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
$nfinite\ ls$
$nfinite\ \sigma$
$nlast\ ls\ = the\text{-}enat(nlength\ \sigma)$
$(\forall\, n\ na.\ na + n \leq nlength\ \sigma \longrightarrow f\ (nsubn\ \sigma\ n\ (n{+}\ na)\ ) \longrightarrow g\ (nsubn\ \sigma\ n\ (n + na)\ ))$
shows
$(\forall\, i{<}nlength\ ls.\ f\ (nsubn\ \sigma\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$\longrightarrow g\ (nsubn\ \sigma\ \ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$)$
using *assms* **unfolding** *nidx-expand* **by** *auto*
(*metis assms(1) diff-add eSuc-enat enat-ord-simps(1) ileI1 le-add-diff-inverse less-imp-le-nat nfinite-nlength-enat nidx-all-le-nlast the-enat.simps*)

lemma *PJ8semhelp-alt*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
$\neg nfinite\ ls$
$\neg nfinite\ \sigma$
$(\forall\, n\ na.\ na + n \leq nlength\ \sigma \longrightarrow f\ (nsubn\ \sigma\ n\ (n{+}\ na)\ ) \longrightarrow g\ (nsubn\ \sigma\ n\ (n + na)\ ))$
shows
$(\forall\, i{<}nlength\ ls.\ f\ (nsubn\ \sigma\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$\longrightarrow g\ (nsubn\ \sigma\ \ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$)$
using *assms* **unfolding** *nidx-expand*
by *auto*
(*metis enat-ord-simps(3) enat-the-enat le-add-diff-inverse nlength-eq-enat-nfiniteD order-less-imp-le*)

lemma *PJ8sem*:
$(\sigma \models ba(f \longrightarrow g) \longrightarrow (f \ \ fproj \ \ h) \ \longrightarrow (g \ \ fproj \ \ h))$
using *PJ8semhelp*
by (*simp add*: *fprojection-d-def ba-defs powerinterval-def*)
(*metis eSuc-enat enat-ord-simps(1) ileI1 le-add-diff-inverse linorder-le-cases nfinite-nlength-enat nidx-expand nidx-less-eq nnth-nlast order-less-imp-le the-enat.simps*)

lemma *OPJ8sem*:
$(\sigma \models ba(f \longrightarrow g) \longrightarrow (f\ oproj\ h) \longrightarrow (g\ oproj\ h))$
using *PJ8semhelp-alt*
by (*simp add*: *oprojection-d-def ba-defs powerinterval-def*)
(*metis eSuc-enat enat-ord-code*(*4*) *enat-the-enat ileI1 le-add-diff-inverse nidx-expand*
nlength-eq-enat-nfiniteD order-less-imp-le order-less-imp-le)

PJ9

lemma *PJ9sem*:
$(\sigma \models f\ ufproj\ (\ g \longrightarrow h) \longrightarrow f\ \ fproj\ \ g \longrightarrow f\ \ fproj\ \ h)$
by (*simp add*: *ufprojection-d-def fprojection-d-def*)
(*metis less-numeral-extra*(*3*))

lemma *OPJ9sem*:
$(\sigma \models f\ uoproj\ (\ g \longrightarrow h) \longrightarrow f\ oproj\ g \longrightarrow f\ oproj\ h)$
by (*simp add*: *uoprojection-d-def oprojection-d-def*)
(*metis less-numeral-extra*(*3*))

3.7.4 Axioms

lemma *FBpGen*:
assumes $\vdash f$
shows $\vdash fbp\ f$
using *assms*
by (*simp add*: *fbp-d-def ufprojection-d-def fprojection-d-def Valid-def*)

lemma *OBpGen*:
assumes $\vdash f$
shows $\vdash obp\ f$
using *assms*
by (*simp add*: *obp-d-def uoprojection-d-def oprojection-d-def Valid-def*)

lemma *PJ00*:
$\vdash \neg(f\ \ fproj\ \ inf)$
using *PJ00sem Valid-def* **by** *blast*

lemma *OPJ00*:
$\vdash \neg(f\ oproj\ finite)$
using *OPJ00sem Valid-def* **by** *blast*

lemma *PJ01*:
$\vdash (f\ \ fproj\ \ g) \longrightarrow finite$
using *PJ01sem Valid-def* **by** *blast*

lemma *OPJ01*:
$\vdash (f\ oproj\ g) \longrightarrow inf$
using *OPJ01sem Valid-def* **by** *blast*

lemma *PJ02*:

$\vdash (f \ \ fproj \ \ g) = ((f \wedge finite) \ \ fproj \ \ g)$
using *PJ02sem Valid-def* **by** *blast*

lemma *OPJ02*:
$\vdash (f \ oproj \ g) = ((f \wedge finite) \ oproj \ g)$
using *OPJ02sem Valid-def* **by** *blast*

lemma *PJ03*:
$\vdash (f \ \ fproj \ \ g) = (f \ \ fproj \ \ (g \wedge finite))$
using *PJ03sem Valid-def* **by** *blast*

lemma *OPJ03*:
$\vdash (f \ oproj \ g) = (f \ oproj \ (g \wedge inf))$
using *OPJ03sem Valid-def* **by** *blast*

lemma *PJ1*:
$\vdash f \ \ fproj \ \ (\ g \vee h) = (f \ \ fproj \ \ g \vee f \ \ fproj \ \ h)$
using *PJ1sem Valid-def* **by** *blast*

lemma *OPJ1*:
$\vdash f \ oproj \ (\ g \vee h) = (f \ oproj \ g \vee f \ oproj \ h)$
using *OPJ1sem Valid-def* **by** *blast*

lemma *PJ2*:
$\vdash f \ \ fproj \ \ empty = empty$
using *PJ2sem Valid-def* **by** *blast*

lemma *OPJ2*:
$\vdash \neg(f \ oproj \ empty)$
using *OPJ2sem Valid-def* **by** *blast*

lemma *PJ3*:
$\vdash f \ \ fproj \ \ skip = (f \wedge more \wedge finite)$
using *PJ3sem Valid-def* **by** *blast*

lemma *OPJ3*:
$\vdash \neg(f \ oproj \ skip)$
using *OPJ3sem Valid-def* **by** *blast*

lemma *PJ4*:
$\vdash f \ \ fproj \ \ (g;h) = (f \ \ fproj \ \ g) \ ; \ (\ f \ \ fproj \ \ h)$
using *PJ4sem Valid-def* **by** *blast*

lemma *OPJ4*:
$\vdash f \ oproj \ ((g \wedge finite);h) = (f \ fproj \ g) \ ; \ (\ f \ oproj \ h)$
using *OPJ4sem Valid-def* **by** *blast*

lemma *PJ5*:
$\vdash f \ \ fproj \ \ init(g) \longrightarrow init(g)$

using *PJ5sem Valid-def* **by** *blast*

lemma *OPJ5*:
$\vdash f\ oproj\ init(g) \longrightarrow init(g)$
using *OPJ5sem Valid-def* **by** *blast*

lemma *PJ6*:
$\vdash skip\ \ fproj\ \ g = (g \wedge finite)$
using *PJ6sem Valid-def* **by** *blast*

lemma *OPJ6*:
$\vdash skip\ oproj\ g = (g \wedge inf)$
using *OPJ6sem Valid-def* **by** *blast*

lemma *PJ7*:
$\vdash f\ \ fproj\ \ (g\ \ fproj\ \ h) = (f\ \ fproj\ \ g)\ \ fproj\ \ h$
using *PJ7sem Valid-def* **by** *blast*

lemma *OPJ7*:
$\vdash f\ oproj\ (g\ oproj\ h) = (f\ \ fproj\ \ g)\ oproj\ h$
using *OPJ7sem Valid-def* **by** *blast*

lemma *PJ8*:
$\vdash ba(f \longrightarrow g) \longrightarrow (f\ \ fproj\ \ h) \longrightarrow (g\ \ fproj\ \ h)$
using *PJ8sem Valid-def* **by** *blast*

lemma *OPJ8*:
$\vdash ba(f \longrightarrow g) \longrightarrow (f\ oproj\ h) \longrightarrow (g\ oproj\ h)$
using *OPJ8sem Valid-def* **by** *blast*

lemma *PJ9*:
$\vdash f\ ufproj\ (g \longrightarrow h) \longrightarrow f\ \ fproj\ \ g \longrightarrow f\ \ fproj\ \ h$
using *PJ9sem Valid-def* **by** *blast*

lemma *OPJ9*:
$\vdash f\ uoproj\ (g \longrightarrow h) \longrightarrow f\ oproj\ g \longrightarrow f\ oproj\ h$
using *OPJ9sem Valid-def* **by** *blast*

3.7.5 Theorems

Projection

lemma *FPowerFProjLen*:
$\vdash f\ \ fproj\ \ len\ n = fpower\ (f \wedge more)\ n$
proof
(*induct n*)
case *0*
then show *?case*
by (*metis PJ2 fpower-d-def len-d-def wpow-0*)
next
case (*Suc n*)

then show *?case*
by (*metis AndMoreAndFiniteEqvAndFmore FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkip-ChopFinite*
FmoreEqvSkipChopFinite PJ3 PJ4sem fpower-d-def intI inteq-reflection len-d-def lift-and-com wpow-Suc)
qed

lemma *FProjLenExist*:
$\vdash f \ \ fproj \ \ (\exists\, n.\ len\ n) = (\exists\ \ n.\ f \ \ fproj \ \ len\ n)$
by (*simp add*: *Valid-def fprojection-d-def*, *blast*)

lemma *FPowerFProjLenExist*:
$\vdash (\exists\ \ n.\ f \ \ fproj \ \ len\ n) = (\exists\ \ n.\ fpower\ (f \wedge more)\ n)$
using *FPowerFProjLen* **by** (*simp add*: *Valid-def FPowerFProjLen*, *blast*)

lemma *RightFProjImpFProj*:
assumes $\vdash g1 \longrightarrow g2$
shows $\vdash f \ \ fproj \ \ g1 \longrightarrow f \ \ fproj \ \ g2$
using *assms*
by (*simp add*: *Valid-def fprojection-d-def*, *blast*)

lemma *RightOProjImpOProj*:
assumes $\vdash g1 \longrightarrow g2$
shows $\vdash f\ oproj\ g1 \longrightarrow f\ oproj\ g2$
using *assms*
by (*simp add*: *Valid-def oprojection-d-def*, *blast*)

lemma *LeftFProjImpFProj*:
assumes $\vdash f1 \longrightarrow f2$
shows $\vdash f1 \ \ fproj \ \ g \longrightarrow f2 \ \ fproj \ \ g$
using *assms*
by (*simp add*: *Valid-def fprojection-d-def powerinterval-def*, *blast*)

lemma *LeftOProjImpOProj*:
assumes $\vdash f1 \longrightarrow f2$
shows $\vdash f1\ oproj\ g \longrightarrow f2\ oproj\ g$
using *assms*
by (*simp add*: *Valid-def oprojection-d-def powerinterval-def*, *blast*)

lemma *RightFProjEqvFProj*:
assumes $\vdash g1 = g2$
shows $\vdash f \ \ fproj \ \ g1 = f \ \ fproj \ \ g2$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *RightOProjEqvOProj*:
assumes $\vdash g1 = g2$
shows $\vdash f\ oproj\ g1 = f\ oproj\ g2$
using *assms*

by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *LeftFProjEqvFProj*:
assumes $\vdash$ *f1* = *f2*
shows $\vdash$ *f1 fproj g* = *f2 fproj g*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *LeftOProjEqvOProj*:
assumes $\vdash$ *f1* = *f2*
shows $\vdash$ *f1 oproj g* = *f2 oproj g*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *FProjTrueEqvSChopstar*:
$\vdash$ *f fproj* #*True* = (*schopstar f*)
proof −
have *1*: $\vdash$ *finite* = ((∃ *n*. *len n*))
by (*simp add*: *Finite-exist-len*)
have *2*: $\vdash$ *f fproj* #*True* = *f fproj* (∃ *n*. *len n*)
using *1*
by (*metis PJ03 Prop03 Prop10 int-simps*(*29*) *inteq-reflection lift-and-com*)
have *3*: $\vdash$ *f fproj* (∃ *n*. *len n*) = (∃ *n*. *fpower* (*f* ∧ *more*) *n*)
using *FPowerFProjLenExist FProjLenExist* **by** *fastforce*
have *4*: $\vdash$ (∃ *n*. *fpower* (*f* ∧ *more*) *n*) = (*schopstar f*)
by (*metis FPowerstardef int-eq schopstar-d-def*)
show *?thesis* **using** *2 3 4* **by** *fastforce*
qed

lemma *OProjTrueEqvAOmega*:
$\vdash$ *f oproj* #*True* = (*aomega f*)
using *infinite-nidx-imp-infinite-interval*
by (*auto simp add*: *Valid-def oprojection-d-def aomega-d-def powerinterval-def*)
(*meson enat-ile linorder-le-cases nfinite-conv-nlength-enat not-le-imp-less*, *blast*)

lemma *OProjTrueEqvOmega*:
$\vdash$ *f oproj* #*True* = (*omega f*)
by (*metis OProjTrueEqvAOmega OmegaEqvAOmega int-eq*)

lemma *FProjSChopstarEqvSChopstarFProj*:
$\vdash$ *f fproj* (*schopstar g*) = (*schopstar* (*f fproj g*))
proof −
have *1*: $\vdash$ *f fproj* (*schopstar g*) = *f fproj* (*g fproj* #*True*)
by (*metis FProjTrueEqvSChopstar RightFProjEqvFProj inteq-reflection*)
have *2*: $\vdash$ *f fproj* (*g fproj* #*True*) = (*f fproj g*) *fproj* #*True*
by (*simp add*: *PJ7*)
have *3*: $\vdash$ (*f fproj g*) *fproj* #*True* = (*schopstar* (*f fproj g*))
by (*simp add*: *FProjTrueEqvSChopstar*)
show *?thesis* **using** *1 2 3* **by** *fastforce*
qed

lemma *OProjOmegaEqvOmegaFProj*:
$\vdash f\ oproj\ (omega\ g) = (omega\ (f\ \ fproj\ \ g))$
proof −
have *1*: $\vdash f\ oproj\ (omega\ g) = f\ oproj\ (g\ oproj\ \#True)$
by (*metis OProjTrueEqvOmega RightOProjEqvOProj inteq-reflection*)
have *2*: $\vdash f\ oproj\ (g\ oproj\ \#True) = (f\ \ fproj\ \ g)\ oproj\ \#True$
by (*simp add*: *OPJ7*)
have *3*: $\vdash (f\ \ fproj\ \ g)\ oproj\ \#True = (omega\ (f\ \ fproj\ \ g))$
by (*simp add*: *OProjTrueEqvOmega*)
show *?thesis* **using** *1 2 3* **by** *fastforce*
qed

lemma *OProjAndImp*:
$\vdash f\ oproj\ (g1 \wedge g2) \longrightarrow f\ oproj\ g1 \wedge f\ oproj\ g2$
by (*meson Prop12 RightOProjImpOProj int-iffD1 lift-and-com*)

lemma *FProjAndImp*:
$\vdash f\ \ fproj\ \ (g1 \wedge g2) \longrightarrow f\ \ fproj\ \ g1 \wedge f\ \ fproj\ \ g2$
by (*meson Prop12 RightFProjImpFProj int-iffD1 lift-and-com*)

lemma *FProjOrDist*:
$\vdash \#True\ \ fproj\ \ (f \vee\ \ g) = (\ \#True\ \ fproj\ \ f\ \vee\ \ \#True\ \ fproj\ \ g)$
using *PJ1* **by** *blast*

lemma *OProjOrDist*:
$\vdash \#True\ oproj\ (f \vee\ \ g) = (\ \#True\ oproj\ f\ \vee\ \ \#True\ oproj\ g)$
using *OPJ1* **by** *blast*

lemma *StateImportFProj*:
$\vdash ((init\ w) \wedge f\ \ fproj\ \ g) = f\ \ fproj\ \ ((init\ w) \wedge g)$
by (*auto simp add*: *Valid-def init-defs fprojection-d-def pfilt-nnth nidx-expand*)
(*metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le*,
metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le)

lemma *StateImportOProj*:
$\vdash ((init\ w) \wedge f\ oproj\ g) = f\ oproj\ ((init\ w) \wedge g)$
by (*auto simp add*: *Valid-def init-defs oprojection-d-def pfilt-nnth nidx-expand*)
(*metis ndropn-0 ndropn-nfirst pfilt-nlength pfilt-nnth zero-enat-def zero-le*,
metis ndropn-0 ndropn-nfirst pfilt-nlength pfilt-nnth zero-enat-def zero-le)

lemma *FProjStateAndNextEqvStateAndMoreChopFProj*:
$\vdash f\ \ fproj\ \ (\ (init\ w) \wedge \bigcirc\ g) = ((init\ w) \wedge (f \wedge more \wedge finite);(f\ fproj\ \ g))$
proof −
have *2*: $\vdash (f \wedge more \wedge finite);(f\ fproj\ \ g) = f\ \ fproj\ \ \bigcirc\ g$
using *PJ3 PJ4* **unfolding** *next-d-def* **by** (*metis inteq-reflection*)
have *3*: $\vdash f\ \ fproj\ \ (\ (init\ w)\ ) \longrightarrow init\ w$
by (*simp add*: *PJ5*)

have *4*: $\vdash$ (*init w* $\wedge$ *f fproj* $\bigcirc$ *g*) = *f fproj* ((*init w*) $\wedge$ $\bigcirc$ *g*)
by (*simp add*: *StateImportFProj*)
have *5*: $\vdash$ *f fproj* ((*init w*) $\wedge$ $\bigcirc$ *g*) $\longrightarrow$ ((*init w*) $\wedge$ (*f* $\wedge$ *more* $\wedge$ *finite*);(*f fproj g*))
using *2 3 FProjAndImp* **by** *fastforce*
from *5 4* **show** *?thesis* **using** *2* **by** *fastforce*
qed

lemma *OProjStateAndNextEqvStateAndMoreChopFProj*:
$\vdash$ *f oproj* ((*init w*) $\wedge$ $\bigcirc$ *g*) = ((*init w*) $\wedge$ (*f* $\wedge$ *more* $\wedge$ *finite*);(*f oproj g*))
proof −
have *1*: $\vdash$ (*skip* $\wedge$ *finite*) = *skip*
using *itl-defs*(*1*) *itl-defs*(*5*) *nlength-eq-enat-nfiniteD* **by** *fastforce*
have *2*: $\vdash$ (*f* $\wedge$ *more* $\wedge$ *finite*);(*f oproj g*) = *f oproj* $\bigcirc$ *g*
using *PJ3 OPJ4* **unfolding** *next-d-def* **by** (*metis 1 inteq-reflection*)
have *3*: $\vdash$ *f oproj* ((*init w*)) $\longrightarrow$ *init w*
by (*simp add*: *OPJ5*)
have *4*: $\vdash$ (*init w* $\wedge$ *f oproj* $\bigcirc$ *g*) = *f oproj* ((*init w*) $\wedge$ $\bigcirc$ *g*)
by (*simp add*: *StateImportOProj*)
have *5*: $\vdash$ *f oproj* ((*init w*) $\wedge$ $\bigcirc$ *g*) $\longrightarrow$ ((*init w*) $\wedge$ (*f* $\wedge$ *more* $\wedge$ *finite*);(*f oproj g*))
using *2 3 OProjAndImp* **by** *fastforce*
from *5 4* **show** *?thesis* **using** *2* **by** *fastforce*
qed

lemma *FProjNext*:
$\vdash$ *f fproj* $\bigcirc$ *g* = (*f* $\wedge$ *more* $\wedge$ *finite*);(*f fproj g*)
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)

lemma *OProjNext*:
$\vdash$ *f oproj* $\bigcirc$ *g* = (*f* $\wedge$ *more* $\wedge$ *finite*);(*f oproj g*)
by (*metis DiamondEmptyEqvFinite DiamondEqvEmptyOrNextDiamond FiniteChopEqvDiamond FiniteChopSkipEqvSkipChopFinite NowImpDiamond OPJ4 PJ3 Prop05 Prop10 SkipChopEqvNext inteq-reflection*)

lemma *FProjWnext*:
$\vdash$ *f fproj* (*wnext g*) = (*empty* $\vee$ (*f* $\wedge$ *more* $\wedge$ *finite*);(*f fproj g*))
proof −
have *1*: $\vdash$ *f fproj* (*wnext g*) = *f fproj* (*empty* $\vee$ $\bigcirc$ *g*)
by (*simp add*: *RightFProjEqvFProj WnextEqvEmptyOrNext*)
have *2*: $\vdash$ *f fproj* (*empty* $\vee$ $\bigcirc$ *g*) = (*empty* $\vee$ *f fproj* ($\bigcirc$ *g*))
using *PJ1 PJ2* **by** *fastforce*
have *3*: $\vdash$ *f fproj* ($\bigcirc$ *g*) = (*f* $\wedge$ *more* $\wedge$ *finite*);(*f fproj g*)
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)
show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *OProjWnext*:
$\vdash$ *f oproj* (*wnext g*) = ((*f* $\wedge$ *more* $\wedge$ *finite*);(*f oproj g*))
proof −

have *1*: $\vdash$ *f oproj* (*wnext g*) = *f oproj* (*empty* $\vee$ $\bigcirc$ *g*)
by (*simp add*: *RightOProjEqvOProj WnextEqvEmptyOrNext*)
have *2*: $\vdash$ *f oproj* (*empty* $\vee$ $\bigcirc$ *g*) = (*f oproj* ($\bigcirc$ *g*))
using *OPJ2*[*of f*] *OPJ1*[*of f LIFT empty LIFT* $\bigcirc$ *g*] **by** *fastforce*
have *3*: $\vdash$ *f oproj* ($\bigcirc$ *g*) = (*f* $\wedge$ *more* $\wedge$ *finite*);(*f oproj g*)
by (*simp add*: *OProjNext*)
show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *FProjIntro*:
assumes $\vdash$ *f* $\longrightarrow$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); *f*
shows $\vdash$ *f* $\wedge$ *finite*$\longrightarrow$ *g fproj* #*True*
using *assms SCSIntro*[*of f g*] *FProjTrueEqvSChopstar*[*of g*] **unfolding** *schop-d-def*
by *fastforce*

lemma *OProjIntro*:
assumes $\vdash$ *f* $\longrightarrow$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); *f*
shows $\vdash$ *f* $\longrightarrow$ *g oproj* #*True*
using *assms OProjTrueEqvOmega*[*of g*]
by (*metis AndMoreAndFiniteEqvAndFmore OmegaIntro inteq-reflection*)

lemma *RightBoxStateImportFProj*:
$\vdash$ $\Box$(*init w*) $\wedge$ *f fproj g* $\longrightarrow$ *f fproj* ($\Box$(*init w*) $\wedge$ *g*)
by (*simp add*: *Valid-def always-defs init-defs fprojection-d-def*)
(*metis ndropn-all ndropn-nfirst nfinite-conv-nlength-enat nle-le pfilt-code*(*1*) *pfilt-nmap-pfilt*)

lemma *RightBoxStateImportOProj*:
$\vdash$ $\Box$(*init w*) $\wedge$ *f oproj g* $\longrightarrow$ *f oproj* ($\Box$(*init w*) $\wedge$ *g*)
by (*simp add*: *Valid-def always-defs init-defs oprojection-d-def*)
(*metis min-def ndropn-nfirst nfinite-conv-nlength-enat nfinite-ntaken ntaken-nlength pfilt-nnth*)

lemma *LeftBoxStateImportFProjhelp*:
(∀ *n*. *enat n* ≤ *nlength wa* ⟶ *w* (*NNil* (*nfirst* (*ndropn n wa*)))) ∧
(∃ *ls*. (∀ *i*. *enat* (*Suc i*) ≤ *nlength ls* ⟶ *nnth ls i* < *nnth ls* (*Suc i*)) ∧
nnth ls 0 = *0* ∧
nfinite ls ∧
nfinite wa ∧
nlast ls = *the-enat* (*nlength wa*) ∧
(∀ *i*. *enat i* < *nlength ls* ⟶ *f* (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*)))) ∧ *g* (*pfilt wa ls*)) ⟶
(∃ *ls*. (∀ *i*. *enat* (*Suc i*) ≤ *nlength ls* ⟶ *nnth ls i* < *nnth ls* (*Suc i*)) ∧
nnth ls 0 = *0* ∧
nfinite ls ∧
nfinite wa ∧
nlast ls = *the-enat* (*nlength wa*) ∧
(∀ *i*. *enat i* < *nlength ls* ⟶
f (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*))) ∧
(∀ *n*. *enat n* ≤ *nlength* (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⟶
w (*NNil* (*nfirst* (*ndropn n* (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*)))))))) ∧

```
        g (pfilt wa ls))
proof
 assume 0: (∀ n. enat n ≤ nlength wa ⟶ w (NNil (nfirst (ndropn n wa)))) ∧
       (∃ ls. (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i)) ∧
            nnth ls 0 = 0 ∧
            nfinite ls ∧
            nfinite wa ∧
            nlast ls = the-enat (nlength wa) ∧
            (∀ i. enat i < nlength ls ⟶ f (nsubn wa (nnth ls i) (nnth ls (Suc i)))) ∧ g (pfilt wa ls))
 show   (∃ ls. (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i)) ∧
            nnth ls 0 = 0 ∧
            nfinite ls ∧
            nfinite wa ∧
            nlast ls = the-enat (nlength wa) ∧
            (∀ i. enat i < nlength ls ⟶
                 f (nsubn wa (nnth ls i) (nnth ls (Suc i))) ∧
                 (∀ n. enat n ≤ nlength (nsubn wa (nnth ls i) (nnth ls (Suc i))) ⟶
                      w (NNil (nfirst (ndropn n (nsubn wa (nnth ls i) (nnth ls (Suc i)))))))) ∧
            g (pfilt wa ls))
  proof −
   have 1: (∀ n. enat n ≤ nlength wa ⟶ w (NNil (nfirst (ndropn n wa))))
    using 0 by auto
   have 2: (∃ ls. (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i)) ∧
            nnth ls 0 = 0 ∧
            nfinite ls ∧
            nfinite wa ∧
            nlast ls = the-enat (nlength wa) ∧
            (∀ i. enat i < nlength ls ⟶
                 f (nsubn wa (nnth ls i) (nnth ls (Suc i)))) ∧ g (pfilt wa ls))
     using 0 by auto
   obtain ls where 3:  (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i)) ∧
            nnth ls 0 = 0 ∧
            nfinite ls ∧
            nfinite wa ∧
            nlast ls = the-enat (nlength wa) ∧
            (∀ i. enat i < nlength ls ⟶
                 f (nsubn wa (nnth ls i) (nnth ls (Suc i)))) ∧ g (pfilt wa ls)
     using 2 by auto
   have 4:  nnth ls 0 = 0
     using 3 by auto
   have 5: (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i))
     using 3 by auto
   have 6:  nlast  ls  = the-enat(nlength wa)
     using 3 by auto
   have 7: (∀ i<nlength ls. f (nsubn wa ( nnth ls i) ( nnth ls (Suc i)) ))
     using 3 by auto
   have 8: g (pfilt wa ls)
     using 3 by auto
   have 9: (∀ i<nlength ls.
            f (nsubn wa ( nnth ls i) ( nnth ls (Suc i))) ∧
```

```
        (∀ n≤(nnth ls (Suc i)) −(nnth ls i).
            w (NNil (nnth wa ((nnth ls i)+n)))))
    using 1 7
    by (metis 0 ndropn-eq-NNil ndropn-nfirst ndropn-nlast nfinite-conv-nlength-enat
        nnth-beyond not-le-imp-less the-enat.simps)
  have 10: (∀ i<nlength ls.
          nlength (nsubn wa ( nnth ls i) ( nnth ls (Suc i)) ) =
           (nnth ls (Suc i)) −(nnth ls i) )
    by (simp add: 3 PJ6help1 Suc-ile-eq nidx-expand)
  have 11: (∀ i<nlength ls.
          (∀ n≤(nnth ls (Suc i)) −(nnth ls i).
             nnth (nsubn wa ( nnth ls i) ( nnth ls (Suc i)) ) n =
             nnth wa ((nnth ls i)+n) ))
    using 3  by (simp add: nsubn-def1 ntaken-nnth)
  have 12: (∀ i<nlength ls.
         f (nsubn wa ( nnth ls i) ( nnth ls (Suc i)) ) ∧
         (∀ n≤nlength (nsubn wa ( nnth ls i) ( nnth ls (Suc i)) ).
            w (NNil (nnth (nsubn wa ( nnth ls i) ( nnth ls (Suc i))) n))))
   using 9 10 11 by simp
  have 13: (∀ i. enat i < nlength ls ⟶
              f (nsubn wa (nnth ls i) (nnth ls (Suc i))) ∧
              (∀ n. enat n ≤ nlength (nsubn wa (nnth ls i) (nnth ls (Suc i))) ⟶
                 w (NNil (nfirst (ndropn n (nsubn wa (nnth ls i) (nnth ls (Suc i))))))))
    by (simp add: 12 ndropn-nfirst)
  show ?thesis
  using 13 3 by blast
  qed
qed

lemma LeftBoxStateImportOProjhelp:
 (∀ n. enat n ≤ nlength wa ⟶ w (NNil (nfirst (ndropn n wa)))) ∧
       (∃ ls. (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i)) ∧
           nnth ls 0 = 0 ∧
           ¬ nfinite ls ∧ ¬ nfinite wa ∧
         (∀ i. enat i < nlength ls ⟶ f (nsubn wa (nnth ls i) (nnth ls (Suc i)))) ∧ g (pfilt wa ls)) ⟶
       (∃ ls. (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i)) ∧
           nnth ls 0 = 0 ∧
           ¬ nfinite ls ∧
           ¬ nfinite wa ∧
           (∀ i. enat i < nlength ls ⟶
              f (nsubn wa (nnth ls i) (nnth ls (Suc i))) ∧
              (∀ n. enat n ≤ nlength (nsubn wa (nnth ls i) (nnth ls (Suc i))) ⟶
                 w (NNil (nfirst (ndropn n (nsubn wa (nnth ls i) (nnth ls (Suc i)))))))) ∧
           g (pfilt wa ls))
proof
 assume 0: (∀ n. enat n ≤ nlength wa ⟶ w (NNil (nfirst (ndropn n wa)))) ∧
       (∃ ls. (∀ i. enat (Suc i) ≤ nlength ls ⟶ nnth ls i < nnth ls (Suc i)) ∧
           nnth ls 0 = 0 ∧
           ¬ nfinite ls ∧ ¬ nfinite wa ∧
         (∀ i. enat i < nlength ls ⟶ f (nsubn wa (nnth ls i) (nnth ls (Suc i)))) ∧ g (pfilt wa ls))
```

show $(\exists\, ls.\ (\forall\, i.\ enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$\neg\ nfinite\ ls \wedge$
$\neg\ nfinite\ wa \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow$
$f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
$(\forall\, n.\ enat\ n \leq nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
$w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))))) \wedge$
$g\ (pfilt\ wa\ ls))$
proof −
have *1*: $(\forall\, n.\ enat\ n \leq nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa))))$
using *0* **by** *auto*
have *2*: $(\exists\, ls.\ (\forall\, i.\ enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$\neg\ nfinite\ ls \wedge \neg\ nfinite\ wa \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls))$
using *0* **by** *auto*
obtain *ls* **where** *3*: $(\forall\, i.\ enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$\neg\ nfinite\ ls \wedge \neg\ nfinite\ wa \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls)$
using *2* **by** *auto*
have *4*: $nnth\ ls\ 0 = 0$
using *3* **by** *auto*
have *5*: $(\forall\, i.\ enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i))$
using *3* **by** *auto*
have *7*: $(\forall\, i{<}nlength\ ls.\ f\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ))$
using *3* **by** *auto*
have *8*: $g\ (pfilt\ wa\ ls)$
using *3* **by** *auto*
have *9*: $(\forall\, i{<}nlength\ ls.$
$f\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))) \wedge$
$(\forall\, n{\leq}(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i).$
$w\ (NNil\ (nnth\ wa\ ((nnth\ ls\ i){+}n)))))$
using *1 7*
by (*metis 0 linorder-le-cases ndropn-eq-NNil ndropn-nfirst nfinite-NNil nfinite-ndropn-b*)
have *10*: $(\forall\, i{<}nlength\ ls.$
$nlength\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ) =$
$(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i)\ )$
by (*simp add*: *3 OPJ6help1 nidx-expand*)
have *11*: $(\forall\, i{<}nlength\ ls.$
$(\forall\, n{\leq}(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i).$
$nnth\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )\ n =$
$nnth\ wa\ ((nnth\ ls\ i){+}n)\ ))$
using *3* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *12*: $(\forall\, i{<}nlength\ ls.$
$f\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ) \wedge$
$(\forall\, n{\leq}nlength\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ).$
$w\ (NNil\ (nnth\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i)))\ n))))$
using *9 10 11* **by** *simp*

have *13*: $(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow$
$f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
$(\forall\, n.\ enat\ n \leq nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
$w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))))$
by (*simp add*: *12 ndropn-nfirst*)
show *?thesis*
using *13 3* **by** *blast*
qed
qed

lemma *LeftBoxStateImportFProj*:
$\vdash \Box(init\ w) \wedge f\ \ fproj\ \ g \longrightarrow (f \wedge \Box\ (init\ w))\ \ fproj\ \ g$
using *LeftBoxStateImportFProjhelp*[*of - w f g*]
by (*simp add*: *Valid-def always-defs init-defs fprojection-d-def powerinterval-def nidx-expand*)

lemma *LeftBoxStateImportOProj*:
$\vdash \Box(init\ w) \wedge f\ oproj\ g \longrightarrow (f \wedge \Box\ (init\ w))\ oproj\ g$
using *LeftBoxStateImportOProjhelp*[*of - w f g*]
by (*simp add*: *Valid-def always-defs init-defs oprojection-d-def powerinterval-def nidx-expand*)

fdp, fbp odp and obp

lemma *NotFDpEqvFBpNot*:
$\vdash (\neg(fdp\ f)) =\ \ fbp\ (\neg\ f)$
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)

lemma *NotODpEqvOBpNot*:
$\vdash (\neg(odp\ f)) =\ \ obp\ (\neg\ f)$
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)

lemma *NotFDBpEqvFDpNot*:
$\vdash (\neg(fbp\ f)) = fdp(\neg\ f)$
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)

lemma *NotODBpEqvODpNot*:
$\vdash (\neg(obp\ f)) = odp(\neg\ f)$
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)

lemma *NowImpFDp*:
$\vdash f \wedge finite \longrightarrow fdp\ f$
proof −
have *1*: $\vdash\ \ (skip \longrightarrow \#True)$
by *simp*
have *2*: $\vdash ba(skip \longrightarrow \#True)$
using *1* **by** (*simp add*: *BaGen*)
have *3*: $\vdash ba(skip \longrightarrow \#True) \longrightarrow (skip\ \ fproj\ \ f \longrightarrow \#True\ \ fproj\ \ f)$
using *PJ8* **by** *blast*
have *4*: $\vdash (skip\ \ fproj\ \ f \longrightarrow \#True\ \ fproj\ \ f)$
using *2 3 MP* **by** *blast*
show *?thesis* **using** *4 PJ6*

by (*metis 4 PJ6 fdp-d-def inteq-reflection*)
qed

lemma *NowImpODp*:
$\vdash f \wedge inf \longrightarrow odp\ f$
proof −
have *1*: $\vdash\ (skip \longrightarrow \#True)$
by *simp*
have *2*: $\vdash ba(skip \longrightarrow \#True)$
using *1* **by** (*simp add*: *BaGen*)
have *3*: $\vdash ba(skip \longrightarrow \#True) \longrightarrow (skip\ oproj\ f \longrightarrow \#True\ oproj\ f)$
using *OPJ8* **by** *blast*
have *4*: $\vdash (skip\ oproj\ f \longrightarrow \#True\ oproj\ f)$
using *2 3 MP* **by** *blast*
show *?thesis* **using** *4 OPJ6* **by** (*metis int-eq odp-d-def*)
qed

lemma *FBpElim*:
$\vdash fbp\ f \wedge finite \longrightarrow f$
proof −
have *1*: $\vdash \neg\ f \wedge finite \longrightarrow fdp\ (\neg\ f)$
by (*simp add*: *NowImpFDp*)
hence *2*: $\vdash \neg(fdp\ (\neg\ f)) \longrightarrow f \vee inf$
unfolding *finite-d-def* **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *Prop13 fbp-d-def fdp-d-def finite-d-def ufprojection-d-def*)
qed

lemma *OBpElim*:
$\vdash obp\ f \wedge inf \longrightarrow f$
proof −
have *1*: $\vdash \neg\ f \wedge inf \longrightarrow odp\ (\neg\ f)$
by (*simp add*: *NowImpODp*)
hence *2*: $\vdash \neg(odp\ (\neg\ f)) \longrightarrow f \vee finite$
unfolding *finite-d-def* **by** *auto*
from *2* **show** *?thesis*
by (*metis InfEqvNotFinite Prop13 inteq-reflection obp-d-def odp-d-def uoprojection-d-def*)
qed

lemma *FBpImpFDpImpFDp*:
$\vdash fbp\ (f \longrightarrow g) \longrightarrow\ fdp\ \ f \longrightarrow\ \ fdp\ \ g$
proof −
have *1*: $\vdash fbp\ (f \longrightarrow g)\ \longrightarrow (\#True\ \ fproj\ \ f) \longrightarrow (\#True\ \ fproj\ \ g)$
by (*simp add*: *PJ9 fbp-d-def*)
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *OBpImpODpImpODp*:
$\vdash obp\ (f \longrightarrow g) \longrightarrow\ \ odp\ \ f \longrightarrow\ \ odp\ \ g$

proof −
have *1*: ⊢ *obp* (*f* ⟶ *g*) ⟶ (#*True oproj f*) ⟶ (#*True oproj g*)
by (*simp add*: *OPJ9 obp-d-def*)
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpContraPosImpDist*:
⊢ *fbp* (¬ *g* ⟶ ¬ *f*) ⟶ (*fbp f*) ⟶ (*fbp g*)
proof −
have *1*: ⊢ *fbp* (¬ *g* ⟶ ¬ *f*) ⟶ (*fdp* (¬ *g*)) ⟶ (*fdp* (¬ *f*))
by (*rule FBpImpFDpImpFDp*)
hence *2*: ⊢ *fbp* (¬ *g* ⟶ ¬ *f*) ⟶ (¬ (*fdp* (¬ *f*))) ⟶ (¬ (*fdp* (¬ *g*))) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
qed

lemma *OBpContraPosImpDist*:
⊢ *obp* (¬ *g* ⟶ ¬ *f*) ⟶ (*obp f*) ⟶ (*obp g*)
proof −
have *1*: ⊢ *obp* (¬ *g* ⟶ ¬ *f*) ⟶ (*odp* (¬ *g*)) ⟶ (*odp* (¬ *f*))
by (*rule OBpImpODpImpODp*)
hence *2*: ⊢ *obp* (¬ *g* ⟶ ¬ *f*) ⟶ (¬ (*odp* (¬ *f*))) ⟶ (¬ (*odp* (¬ *g*))) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
qed

lemma *FBpImpDist*:
⊢ *fbp* (*f* ⟶ *g*) ⟶ (*fbp f*) ⟶ (*fbp g*)
proof −
have *1*: ⊢ (*f* ⟶ *g*) ⟶ (¬ *g* ⟶ ¬ *f*) **by** *auto*
hence *2*: ⊢ ¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*) **by** *auto*
hence *3*: ⊢ *fbp* (¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*)) **by** (*rule FBpGen*)
have *4*: ⊢ *fbp* (¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*))
⟶
fbp (*f* ⟶ *g*) ⟶ *fbp* (¬ *g* ⟶ ¬ *f*) **by** (*rule FBpContraPosImpDist*)
have *5*: ⊢ *fbp* (*f* ⟶ *g*) ⟶ *fbp* (¬ *g* ⟶ ¬ *f*) **using** *3 4 MP* **by** *blast*
have *6*: ⊢ *fbp* (¬ *g* ⟶ ¬ *f*) ⟶ (*fbp f*) ⟶ (*fbp g*) **by** (*rule FBpContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *OBpImpDist*:
⊢ *obp* (*f* ⟶ *g*) ⟶ (*obp f*) ⟶ (*obp g*)
proof −
have *1*: ⊢ (*f* ⟶ *g*) ⟶ (¬ *g* ⟶ ¬ *f*) **by** *auto*
hence *2*: ⊢ ¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*) **by** *auto*
hence *3*: ⊢ *obp* (¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*)) **by** (*rule OBpGen*)
have *4*: ⊢ *obp* (¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*))
⟶
obp (*f* ⟶ *g*) ⟶ *obp* (¬ *g* ⟶ ¬ *f*) **by** (*rule OBpContraPosImpDist*)
have *5*: ⊢ *obp* (*f* ⟶ *g*) ⟶ *obp* (¬ *g* ⟶ ¬ *f*) **using** *3 4 MP* **by** *blast*

have *6*: $\vdash$ *obp* (¬ *g* ⟶ ¬ *f*) ⟶ (*obp f*) ⟶ (*obp g*) **by** (*rule OBpContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FDpImpDpRule*:
assumes $\vdash$ *f* ⟶ *g*
shows $\vdash$ *fdp f* ⟶ *fdp g*
proof −
have *1*: $\vdash$ *f* ⟶ *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ #*True fproj f* ⟶ #*True fproj g*
by (*metis FBpGen MP PJ9 fbp-d-def*)
from *2* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpImpODpRule*:
assumes $\vdash$ *f* ⟶ *g*
shows $\vdash$ *odp f* ⟶ *odp g*
proof −
have *1*: $\vdash$ *f* ⟶ *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ #*True oproj f* ⟶ #*True oproj g*
by (*metis OBpGen MP OPJ9 obp-d-def*)
from *2* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpImpFBpRule*:
assumes $\vdash$ *f* ⟶ *g*
shows $\vdash$ *fbp f* ⟶ *fbp g*
proof −
have *1*: $\vdash$ *f* ⟶ *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ ¬ *g* ⟶ ¬ *f* **by** *auto*
hence *3*: $\vdash$ *fdp* (¬ *g*) ⟶ *fdp* (¬ *f*) **by** (*rule FDpImpDpRule*)
hence *4*: $\vdash$ ¬ (*fdp* (¬ *f*)) ⟶ ¬ (*fdp* (¬ *g*)) **by** *auto*
from *4* **show** *?thesis*
by (*meson FBpGen FBpImpDist MP assms*)
qed

lemma *OBpImpOBpRule*:
assumes $\vdash$ *f* ⟶ *g*
shows $\vdash$ *obp f* ⟶ *obp g*
proof −
have *1*: $\vdash$ *f* ⟶ *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ ¬ *g* ⟶ ¬ *f* **by** *auto*
hence *3*: $\vdash$ *odp* (¬ *g*) ⟶ *odp* (¬ *f*) **by** (*rule ODpImpODpRule*)
hence *4*: $\vdash$ ¬ (*odp* (¬ *f*)) ⟶ ¬ (*odp* (¬ *g*)) **by** *auto*
from *4* **show** *?thesis*
by (*meson OBpGen OBpImpDist MP assms*)
qed

lemma *FDpEqvFDpRule*:
assumes $\vdash$ *f* = *g*

shows $\vdash$ *fdp f* = *fdp g*
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ #*True fproj f* = #*True fproj g*
using *RightFProjEqvFProj* **by** *blast*
from *2* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpEqvODpRule*:
assumes $\vdash f = g$
shows $\vdash$ *odp f* = *odp g*
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ #*True oproj f* = #*True oproj g*
using *RightOProjEqvOProj* **by** *blast*
from *2* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpEqvFBpRule*:
assumes $\vdash f = g$
shows $\vdash$ *fbp f* = *fbp g*
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ f) = (\neg\ g)$ **by** *auto*
hence *3*: $\vdash$ *fdp* $(\neg\ f)$ = *fdp* $(\neg\ g)$ **by** (*rule FDpEqvFDpRule*)
hence *4*: $\vdash (\neg\ (fdp\ (\neg\ f))) = (\neg\ (\ fdp\ (\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis*
by (*metis FBpImpFBpRule assms int-iffD1 int-iffI inteq-reflection*)
qed

lemma *OBpEqvOBpRule*:
assumes $\vdash f = g$
shows $\vdash$ *obp f* = *obp g*
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ f) = (\neg\ g)$ **by** *auto*
hence *3*: $\vdash$ *odp* $(\neg\ f)$ = *odp* $(\neg\ g)$ **by** (*rule ODpEqvODpRule*)
hence *4*: $\vdash (\neg\ (odp\ (\neg\ f))) = (\neg\ (\ odp\ (\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis*
by (*metis OBpImpOBpRule assms int-iffD1 int-iffI inteq-reflection*)
qed

lemma *FDpState*:
$\vdash$ *fdp* (*init w*) = ((*init w*) $\wedge$ *finite*)
proof −
have *1*: $\vdash$ *init w* $\wedge$ *finite* $\longrightarrow$ *fdp* (*init w*)
using *NowImpFDp*[*of LIFT* (*init w*)] **by** *blast*
have *2*: $\vdash$*fdp* (*init w*) $\longrightarrow$ *init w*
unfolding *fdp-d-def* **by** (*simp add*: *PJ5*)
have *3*: $\vdash$*fdp* (*init w*) $\longrightarrow$ *finite*

unfolding *fdp-d-def* **by** (*simp add*: *PJ01*)
show *?thesis*
by (*simp add*: *1 2 3 Prop12 int-iffI*)
qed

lemma *ODpState*:
$\vdash$ *odp* (*init w*) = ((*init w*) $\wedge$ *inf*)
proof −
have *1*: $\vdash$ *init w* $\wedge$ *inf* $\longrightarrow$ *odp* (*init w*)
using *NowImpODp*[*of LIFT* (*init w*)] **by** *blast*
have *2*: $\vdash$*odp* (*init w*) $\longrightarrow$ *init w*
unfolding *odp-d-def* **by** (*simp add*: *OPJ5*)
have *3*: $\vdash$*odp* (*init w*) $\longrightarrow$ *inf*
unfolding *odp-d-def* **by** (*simp add*: *OPJ01*)
show *?thesis*
by (*simp add*: *1 2 3 Prop12 int-iffI*)
qed

lemma *StateEqvFBp*:
$\vdash$ *finite* $\longrightarrow$ (*init w*) = *fbp* (*init w*)
proof −
have *1*: $\vdash$ (*init w*) $\longrightarrow$ *fbp* (*init w*)
by (*metis* (*no-types*, *lifting*) *DiEqvNotBiNot DiState Initprop*(*2*) *NotDiEqvBiNot PJ5 fbp-d-def*
inteq-reflection lift-imp-neg ufprojection-d-def)
have *2*: $\vdash$ *fbp* (*init w*) $\wedge$ *finite* $\longrightarrow$ (*init w*) **using** *FBpElim* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *StateEqvOBp*:
$\vdash$ *inf* $\longrightarrow$ (*init w*) = *obp* (*init w*)
proof −
have *1*: $\vdash$ (*init w*) $\longrightarrow$ *obp* (*init w*)
by (*metis* (*no-types*, *lifting*) *DiEqvNotBiNot DiState Initprop*(*2*) *NotDiEqvBiNot NotODpEqvOBpNot*
ODpState Prop11 Prop12 inteq-reflection lift-imp-neg)
have *2*: $\vdash$ *obp* (*init w*) $\wedge$ *inf* $\longrightarrow$ (*init w*) **using** *OBpElim* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FDpFDpEqvFDp*:
$\vdash$ *fdp* (*fdp f*) = *fdp f*
proof −
have *2*: $\vdash$ #*True fproj* (#*True fproj f*) = (#*True fproj* #*True*) *fproj f*
by (*simp add*: *PJ7*)
have *3*: $\vdash$ (#*True fproj* #*True*) = *finite*
by (*metis ChopEmpty EmptyImpFinite NowImpFDp PJ01 Prop10 TrueChopAndFiniteEqvAndFinite-*
ChopFinite
fdp-d-def int-eq int-iffI)
have *4*: $\vdash$ *finite fproj f* = #*True fproj f*
by (*metis PJ02 Prop03 Prop10 finite-d-def int-simps*(*28*) *inteq-reflection lift-and-com*)
show *?thesis*

by (*metis 2 3 4 fdp-d-def int-eq*)
qed

lemma *ODpODpEqvODp*:
$\vdash$ *odp* (*odp f*) = *odp f*
proof −
have *2*: $\vdash$ #*True oproj* (#*True oproj f*) = (#*True fproj* #*True*) *oproj f*
by (*simp add*: *OPJ7*)
have *3*: $\vdash$ (#*True fproj* #*True*) = *finite*
by (*metis ChopEmpty EmptyImpFinite NowImpFDp PJ01 Prop10 TrueChopAndFiniteEqvAndFinite-ChopFinite*
fdp-d-def int-eq int-iffI)
have *4*: $\vdash$ *finite oproj f* = #*True oproj f*
by (*metis OPJ02 Prop03 Prop10 finite-d-def int-simps*(*28*) *inteq-reflection lift-and-com*)
show *?thesis*
by (*metis 2 3 4 odp-d-def int-eq*)
qed

lemma *FBpFBpEqvFBp*:
$\vdash$ *fbp* (*fbp f*) = *fbp f*
proof −
have *1*: $\vdash$ *fdp* (*fdp* (¬ *f*)) = *fdp* (¬ *f*)
using *FDpFDpEqvFDp* **by** *blast*
have *2*: $\vdash$ (¬ (*fdp* (*fdp* (¬ *f*)))) = (¬ (*fdp* (¬ *f*)))
using *1* **by** *auto*
have *3*: $\vdash$ (¬ (*fdp* (¬ *f*))) = *fbp f*
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
have *4*: $\vdash$ (¬ (*fdp* (*fdp* (¬ *f*)))) = *fbp* (*fbp f*)
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
from *2 3 4* **show** *?thesis*
by *fastforce*
qed

lemma *OBpOBpEqvOBp*:
$\vdash$ *obp* (*obp f*) = *obp f*
proof −
have *1*: $\vdash$ *odp* (*odp* (¬ *f*)) = *odp* (¬ *f*)
using *ODpODpEqvODp* **by** *blast*
have *2*: $\vdash$ (¬ (*odp* (*odp* (¬ *f*)))) = (¬ (*odp* (¬ *f*)))
using *1* **by** *auto*
have *3*: $\vdash$ (¬ (*odp* (¬ *f*))) = *obp f*
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
have *4*: $\vdash$ (¬ (*odp* (*odp* (¬ *f*)))) = *obp* (*obp f*)
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
from *2 3 4* **show** *?thesis*
by *fastforce*
qed

lemma *FDpOrEqv*:

$\vdash$ *fdp* (*f* $\vee$ *g*) = (*fdp f* $\vee$ *fdp g*)
proof −
have *1*: $\vdash$ #*True fproj* (*f*$\vee$ *g*) = (#*True fproj f* $\vee$ #*True fproj g*)
using *FProjOrDist* **by** *auto*
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpOrEqv*:
$\vdash$ *odp* (*f* $\vee$ *g*) = (*odp f* $\vee$ *odp g*)
proof −
have *1*: $\vdash$ #*True oproj* (*f* $\vee$ *g*) = (#*True oproj f* $\vee$ #*True oproj g*)
using *OProjOrDist* **by** *auto*
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpAndEqv*:
$\vdash$ *fbp*(*f* $\wedge$ *g*) = (*fbp f* $\wedge$ *fbp g*)
proof −
have *1*: $\vdash$ *fdp* ((¬ *f*) $\vee$ (¬*g*)) = (*fdp* (¬*f*) $\vee$ *fdp* (¬*g*))
using *FDpOrEqv* **by** *auto*
hence *2*: $\vdash$ (¬ (*fdp* ((¬ *f*) $\vee$ (¬*g*)))) = (¬(*fdp* (¬*f*) $\vee$ *fdp* (¬*g*)))
by *auto*
have *3*: $\vdash$ (¬ (*fdp* ((¬ *f*) $\vee$ (¬*g*)))) = *fbp* (¬((¬ *f*) $\vee$ (¬*g*)))
using *NotFDpEqvFBpNot* **by** *blast*
have *4*: $\vdash$ (¬((¬ *f*) $\vee$ (¬*g*))) =(*f* $\wedge$ *g*)
by *auto*
hence *5*: $\vdash$ *fbp*(¬((¬ *f*) $\vee$ (¬*g*))) = *fbp*(*f* $\wedge$ *g*)
by (*simp add*: *FBpEqvFBpRule*)
have *6*: $\vdash$ (¬(*fdp* (¬*f*) $\vee$ *fdp* (¬*g*))) = ((¬(*fdp* (¬ *f*))) $\wedge$ (¬(*fdp* (¬ *g*))))
by *auto*
have *7*: $\vdash$ ((¬(*fdp* (¬ *f*))) $\wedge$ (¬(*fdp* (¬ *g*)))) = (*fbp f* $\wedge$ *fbp g*)
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
show *?thesis*
by (*metis 2 3 4 6 7 inteq-reflection*)
qed

lemma *OBpAndEqv*:
$\vdash$ *obp*(*f* $\wedge$ *g*) = (*obp f* $\wedge$ *obp g*)
proof −
have *1*: $\vdash$ *odp* ((¬ *f*) $\vee$ (¬*g*)) = (*odp* (¬*f*) $\vee$ *odp* (¬*g*))
using *ODpOrEqv* **by** *auto*
hence *2*: $\vdash$ (¬ (*odp* ((¬ *f*) $\vee$ (¬*g*)))) = (¬(*odp* (¬*f*) $\vee$ *odp* (¬*g*)))
by *auto*
have *3*: $\vdash$ (¬ (*odp* ((¬ *f*) $\vee$ (¬*g*)))) = *obp* (¬((¬ *f*) $\vee$ (¬*g*)))
using *NotODpEqvOBpNot* **by** *blast*
have *4*: $\vdash$ (¬((¬ *f*) $\vee$ (¬*g*))) =(*f* $\wedge$ *g*)
by *auto*
hence *5*: $\vdash$ *obp*(¬((¬ *f*) $\vee$ (¬*g*))) = *obp*(*f* $\wedge$ *g*)
by (*simp add*: *OBpEqvOBpRule*)
have *6*: $\vdash$ (¬(*odp* (¬*f*) $\vee$ *odp* (¬*g*))) = ((¬(*odp* (¬ *f*))) $\wedge$ (¬(*odp* (¬ *g*))))

by *auto*
have *7*: $\vdash ((\neg(odp\ (\neg\ f))) \wedge (\neg(odp\ (\neg\ g)))) = (obp\ f \wedge obp\ g)$
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
show *?thesis*
by (*metis 2 3 4 6 7 inteq-reflection*)
qed

lemma *FDpAndA*:
$\vdash \quad fdp\ (f \wedge g) \longrightarrow \ fdp\ \ f$
proof −
have *1*: $\vdash \#True\ \ fproj\ \ (f \wedge g) \longrightarrow \#True\ \ fproj\ \ f$
by (*meson Prop12 RightFProjImpFProj int-iffD1 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpAndA*:
$\vdash \quad odp\ (f \wedge g) \longrightarrow \ odp\ \ f$
proof −
have *1*: $\vdash \#True\ oproj\ (f \wedge g) \longrightarrow \#True\ oproj\ f$
by (*meson Prop12 RightOProjImpOProj int-iffD1 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpOrA*:
$\vdash fbp\ f \longrightarrow \ fbp(f \vee g)$
by (*simp add*: *FBpImpFBpRule intI*)

lemma *OBpOrA*:
$\vdash obp\ f \longrightarrow \ obp(f \vee g)$
by (*simp add*: *OBpImpOBpRule intI*)

lemma *FBpOrB*:
$\vdash fbp\ g \longrightarrow \ fbp(f \vee g)$
by (*simp add*: *FBpImpFBpRule intI*)

lemma *OBpOrB*:
$\vdash obp\ g \longrightarrow \ obp(f \vee g)$
by (*simp add*: *OBpImpOBpRule intI*)

lemma *FBpOrImpOr*:
$\vdash fbp\ f \vee fbp\ g \longrightarrow fbp(f \vee g)$
using *FBpOrA FBpOrB* **by** *fastforce*

lemma *OBpOrImpOr*:
$\vdash obp\ f \vee obp\ g \longrightarrow obp(f \vee g)$
using *OBpOrA OBpOrB* **by** *fastforce*

lemma *FDpAndB*:
$\vdash \quad fdp\ (f \wedge g) \longrightarrow \ fdp\ \ g$
proof −

have *1*: $\vdash \#True\ \ fproj\ \ (f \wedge g) \longrightarrow \#True\ \ fproj\ \ g$
by (*meson Prop12 RightFProjImpFProj int-iffD2 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpAndB*:
$\vdash \quad odp\ (f \wedge g) \longrightarrow\ \ odp\ \ g$
proof −
have *1*: $\vdash \#True\ oproj\ (f \wedge g) \longrightarrow \#True\ oproj\ g$
by (*meson Prop12 RightOProjImpOProj int-iffD2 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FDpAndImpAnd*:
$\vdash \quad fdp\ (f \wedge g) \longrightarrow\ \ fdp\ \ f \wedge\ \ fdp\ \ g$
proof −
have *1*: $\vdash\ \ fdp\ (f \wedge g) \longrightarrow\ \ fdp\ \ f$ **by** (*rule FDpAndA*)
have *2*: $\vdash\ \ fdp\ (f \wedge g) \longrightarrow\ \ fdp\ \ g$ **by** (*rule FDpAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ODpAndImpAnd*:
$\vdash \quad odp\ (f \wedge g) \longrightarrow\ \ odp\ \ f \wedge\ \ odp\ \ g$
proof −
have *1*: $\vdash\ \ odp\ (f \wedge g) \longrightarrow\ \ odp\ \ f$ **by** (*rule ODpAndA*)
have *2*: $\vdash\ \ odp\ (f \wedge g) \longrightarrow\ \ odp\ \ g$ **by** (*rule ODpAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FDpSkipEqvMore*:
$\vdash \quad fdp\ \ skip\ \ =\ \ (more \wedge finite)$
proof −
have *1*: $\vdash fdp\ skip = \#True\ \ fproj\ \ skip$
by (*simp add*: *fdp-d-def*)
have *2*: $\vdash \#True\ \ fproj\ \ skip = (\#True \wedge more \wedge finite)$
using *PJ3* **by** *blast*
have *3*: $\vdash (\#True \wedge more\ ) = more$
by *auto*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FDpMoreEqvMore*:
$\vdash \quad fdp\ \ more\ \ =\ \ (more \wedge finite)$
using *FDpFDpEqvFDp FDpSkipEqvMore*
by (*metis PJ03 fdp-d-def inteq-reflection*)

lemma *ODpMoreEqvInf*:
$\vdash \quad odp\ \ more\ \ =\ \ (\ inf)$
by (*metis MoreAndInfEqvInf NowImpODp OPJ01 int-iffI inteq-reflection odp-d-def*)

lemma *FBpEmptyEqvEmpty*:
$\vdash$ *fbp empty* = (*empty* $\vee$ *inf*)
proof −
have *1*: $\vdash$ *fbp empty* = ($\neg$ (*fdp more*))
by (*metis NotFDpEqvFBpNot Prop11 empty-d-def*)
have *2*: $\vdash$ ($\neg$ (*fdp more*)) = ($\neg$ (*more* $\wedge$ *finite*))
using *FDpMoreEqvMore* **by** *auto*
have *3*: $\vdash$ ($\neg$ (*more* $\wedge$ *finite*)) = (*empty* $\vee$ *inf*)
unfolding *finite-d-def empty-d-def* **by** *fastforce*
show *?thesis*
by (*metis 1 2 3 int-eq*)
qed

lemma *OBpEmptyEqvFinite*:
$\vdash$ *obp empty* = (*finite*)
proof −
have *1*: $\vdash$ *obp empty* = ($\neg$ (*odp more*))
by (*metis NotODpEqvOBpNot Prop11 empty-d-def*)
have *2*: $\vdash$ ($\neg$ (*odp more*)) = ($\neg$ (*inf*))
using *ODpMoreEqvInf* **by** *auto*
have *3*: $\vdash$ ($\neg$ (*inf*)) = (*finite*)
unfolding *finite-d-def* **by** *fastforce*
show *?thesis*
by (*metis 1 2 3 int-eq*)
qed

lemma *FDpEmptyEqvEmpty*:
$\vdash$ *fdp empty* = *empty*
proof −
have *1*: $\vdash$ *fdp empty* = #*True fproj empty*
by (*simp add*: *fdp-d-def*)
have *2*: $\vdash$ #*True fproj empty* = *empty*
by (*simp add*: *PJ2*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotODpEmpty*:
$\vdash$ $\neg$(*odp empty*)
proof −
have *1*: $\vdash$ *odp empty* = #*True oproj empty*
by (*simp add*: *odp-d-def*)
show *?thesis*
by (*metis 1 OPJ2 int-eq*)
qed

lemma *FBpMoreEqvMore*:
$\vdash$ *fbp more* = *more*
by (*metis NotFDBpEqvFDpNot PJ2 empty-d-def fdp-d-def int-eq int-simps*(*4*))

lemma *OBpMore*:
$\vdash$ *obp more*
by (*metis OPJ2 empty-d-def obp-d-def uoprojection-d-def*)

lemma *NextFDpImpFDpNext*:
$\vdash \bigcirc (fdp\ f) \longrightarrow fdp\ (\bigcirc f)$
proof −
have *1*: $\vdash fdp(\bigcirc f) = \#True\ \ fproj\ \ (skip;f)$
by (*simp add*: *fdp-d-def next-d-def*)
have *2*: $\vdash \#True\ \ fproj\ \ (skip;f) = (\#True\ \ fproj\ \ skip);(\#True\ \ fproj\ \ f)$
by (*simp add*: *PJ4*)
have *3*: $\vdash (\#True\ \ fproj\ \ skip) = (\#True \wedge more \wedge finite)$
using *PJ3* **by** *blast*
have *4*: $\vdash (\#True \wedge more) = more$
by *auto*
have *40*: $\vdash skip \longrightarrow more$
by (*metis DiIntro DiSkipEqvMore int-eq*)
have *41*: $\vdash skip \longrightarrow finite$
by (*metis AndChopB EmptySChop FiniteChopSkipImpFinite Prop11 lift-imp-trans schop-d-def*)
have *5*: $\vdash skip;(\#True\ \ fproj\ \ f) \longrightarrow (more \wedge finite);(\#True\ \ fproj\ \ f)$
by (*simp add*: *40 41 LeftChopImpChop Prop12*)
show *?thesis* **by** (*metis 2 5 FDpSkipEqvMore fdp-d-def inteq-reflection next-d-def*)
qed

lemma *NextODpImpODpNext*:
$\vdash \bigcirc (odp\ f) \longrightarrow odp\ (\bigcirc f)$
proof −
have *1*: $\vdash odp(\bigcirc f) = \#True\ oproj\ (skip;f)$
by (*simp add*: *odp-d-def next-d-def*)
have *10*: $\vdash (skip \wedge finite) = skip$
by (*metis FiniteChopEqvDiamond FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 Prop11 SkipChopFiniteImpFinite lift-imp-trans*)
have *2*: $\vdash \#True\ oproj\ (skip;f) = (\#True\ \ fproj\ \ skip);(\#True\ oproj\ f)$
using *OPJ4* **by** (*metis 10 inteq-reflection*)
have *3*: $\vdash (\#True\ \ fproj\ \ skip) = (\#True \wedge more \wedge finite)$
using *PJ3* **by** *blast*
have *4*: $\vdash (\#True \wedge more) = more$
by *auto*
have *40*: $\vdash skip \longrightarrow more$
by (*metis DiIntro DiSkipEqvMore int-eq*)
have *41*: $\vdash skip \longrightarrow finite$
by (*metis AndChopB EmptySChop FiniteChopSkipImpFinite Prop11 lift-imp-trans schop-d-def*)
have *5*: $\vdash skip;(\#True\ oproj\ f) \longrightarrow (more \wedge finite);(\#True\ oproj\ f)$
by (*simp add*: *40 41 LeftChopImpChop Prop12*)
show *?thesis* **by** (*metis 2 5 FDpSkipEqvMore fdp-d-def inteq-reflection next-d-def odp-d-def*)
qed

lemma *BoxStateImportFBp*:

$\vdash \Box(init\ w) \longrightarrow fbp\ (\Box(init\ w))$
proof −
have *1*: $\vdash fbp\ (\Box(init\ w)) = (\neg(fdp\ (\Diamond\ (\neg(init\ w)))))$
by (*metis NotFDpEqvFBpNot always-d-def int-eq*)
have *2*: $\vdash\ (\Diamond\ (\neg(init\ w))) = (finite;(\neg(init\ w)))$
by (*simp add*: *sometimes-d-def*)
have *3*: $\vdash fdp\ (finite;(\neg(init\ w))) = (fdp\ finite);\ (fdp\ (\neg(init\ w)))$
by (*simp add*: *PJ4 fdp-d-def*)
have *4*: $\vdash (fdp\ finite) = finite$
by (*metis EmptyOrMoreSplit FDpEmptyEqvEmpty FiniteAndEmptyEqvEmpty FiniteChopSkipEqvFinite-AndMore*
FiniteChopSkipImpFinite NowImpFDp PJ01 Prop02 Prop10 fdp-d-def int-iffI inteq-reflection)
have *5*: $\vdash (fdp\ (\neg(init\ w))) = ((\neg(init\ w)) \wedge finite)$
by (*metis FDpState Initprop*(*2*) *inteq-reflection*)
have *6*: $\vdash finite;((\neg(init\ w)) \wedge finite) \longrightarrow \Diamond(\neg(init\ w))$
by (*metis 2 ChopAndA inteq-reflection*)
show *?thesis*
by (*metis 1 2 3 4 5 6 always-d-def inteq-reflection lift-imp-neg*)
qed

lemma *BoxStateImportOBp*:
$\vdash \Box(init\ w) \longrightarrow obp\ (\Box(init\ w))$
proof −
have *1*: $\vdash obp\ (\Box(init\ w)) = (\neg(odp\ (\Diamond\ (\neg(init\ w)))))$
by (*metis NotODpEqvOBpNot always-d-def int-eq*)
have *2*: $\vdash\ (\Diamond\ (\neg(init\ w))) = (finite;(\neg(init\ w)))$
by (*simp add*: *sometimes-d-def*)
have *3*: $\vdash odp\ (finite;(\neg(init\ w))) = (fdp\ finite);\ (odp\ (\neg(init\ w)))$
unfolding *odp-d-def fdp-d-def*
using *OPJ4*[*of LIFT* # *True LIFT finite LIFT* $(\neg(init\ w))$]
by (*simp add*: *OPJ4 odp-d-def fdp-d-def*)
have *4*: $\vdash (fdp\ finite) = finite$
by (*metis EmptyOrMoreSplit FDpEmptyEqvEmpty FiniteAndEmptyEqvEmpty FiniteChopSkipEqvFinite-AndMore*
FiniteChopSkipImpFinite NowImpFDp PJ01 Prop02 Prop10 fdp-d-def int-iffI inteq-reflection)
have *5*: $\vdash (odp\ (\neg(init\ w))) = ((\neg(init\ w)) \wedge inf)$
by (*metis ODpState Initprop*(*2*) *inteq-reflection*)
have *6*: $\vdash finite;((\neg(init\ w)) \wedge inf) \longrightarrow \Diamond(\neg(init\ w))$
by (*metis 2 ChopAndA inteq-reflection*)
show *?thesis* **by** (*metis 1 2 3 4 5 6 always-d-def int-eq lift-imp-neg*)
qed

lemma *BoxStateEqvFBpBoxState*:
$\vdash finite \longrightarrow \Box\ (init\ w) = fbp(\Box\ (init\ w))$
proof −
have *1*: $\vdash\ finite \longrightarrow fbp(\Box\ (init\ w)) \longrightarrow \Box\ (init\ w)$
by (*metis FBpElim Prop09 inteq-reflection lift-and-com*)
have *2*: $\vdash\ fbp(\Box\ (init\ w)) = (\neg(\#True\ \ fproj\ \ (\neg\ \Box\ (init\ w))))$
by (*simp add*: *fbp-d-def ufprojection-d-def*)
have *2*: $\vdash \Box(init\ w) \wedge finite \longrightarrow fdp\ (\Box(init\ w))$

by (*metis NowImpFDp*)
have *2*: $\vdash \Box\ (init\ w) \longrightarrow fbp(\Box\ (init\ w))$
using *BoxStateImportFBp* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvOBpBoxState*:
$\vdash inf \longrightarrow \Box\ (init\ w) = obp(\Box\ (init\ w))$
proof −
have *1*: $\vdash\ inf \longrightarrow obp(\Box\ (init\ w)) \longrightarrow \Box\ (init\ w)$
by (*metis OBpElim Prop09 inteq-reflection lift-and-com*)
have *2*: $\vdash\ obp(\Box\ (init\ w)) = (\neg(\#True\ oproj\ (\neg\ \Box\ (init\ w))))$
by (*simp add*: *obp-d-def uoprojection-d-def*)
have *2*: $\vdash \Box(init\ w) \wedge inf \longrightarrow odp\ (\Box(init\ w))$
by (*metis NowImpODp*)
have *2*: $\vdash \Box\ (init\ w) \longrightarrow obp(\Box\ (init\ w))$
using *BoxStateImportOBp* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

end

3.8 Until operator

theory *Until*
imports *Semantics SChopTheorems*
begin

This theory introduces the weak and strong versions of the until operator. The theorems from [18] are proven in a mostly deductive style.

3.8.1 Definitions

definition *until-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where *until-d F G* $\equiv \lambda s.\ (\ (\exists\ k\ .\ k \leq nlength\ s \wedge (\ (ndropn\ k\ s\ ) \models G) \wedge$
$(\forall j.\ j < k\ \longrightarrow\ (\ (ndropn\ j\ s\ ) \models F)\ ))\ )$

syntax
-until-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{U}$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-until-d :: [*lift*,*lift*] ⇒ *lift* ((- *until* -) [*84*,*84*] *83*)

translations
-until-d ⇌ *CONST until-d*

definition *suntil-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where *suntil-d F G* $\equiv LIFT(\bigcirc(F\ \mathcal{U}\ G))$

definition *wait-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$

where *wait-d F G* ≡ *LIFT*(□ *F* ∨ *F* $\mathcal{U}$ *G*)

definition *release-d* :: ('*a* :: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *release-d F G* ≡ *LIFT*(¬((¬ *F*) $\mathcal{U}$ (¬ *G*)))

syntax
-wait-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{W}$ -) [*84*,*84*] *83*)
-release-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{R}$ -) [*84*,*84*] *83*)
-suntil-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{U}^s$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-wait-d :: [*lift*,*lift*] ⇒ *lift* ((- *wait* -) [*84*,*84*] *83*)
-release-d :: [*lift*,*lift*] ⇒ *lift* ((- *release* -) [*84*,*84*] *83*)
-suntil-d :: [*lift*,*lift*] ⇒ *lift* ((- *suntil* -) [*84*,*84*] *83*)

translations
-wait-d ⇌ *CONST wait-d*
-release-d ⇌ *CONST release-d*
-suntil-d ⇌ *CONST suntil-d*

definition *srelease-d* :: ('*a* :: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *srelease-d F G* ≡ *LIFT*(¬((¬ *F*) $\mathcal{W}$ (¬ *G*)))

syntax
-srelease-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{M}$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-srelease-d :: [*lift*,*lift*] ⇒ *lift* ((- *srelease* -) [*84*,*84*] *83*)

translations
-srelease-d ⇌ *CONST srelease-d*

definition *since-d* :: ('*a* :: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *since-d F G* ≡
λ*s*. ((∃ *k* . *k* ≤ *nlength s* ∧ ((*ntaken k s*) ⊨ *G*) ∧
(∀ *j*. *k*< *j* ∧ *j* ≤ *nlength s* ⟶ ((*ntaken j s*) ⊨ *F*))))

syntax
-since-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{S}$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-since-d :: [*lift*,*lift*] ⇒ *lift* ((- *since* -) [*84*,*84*] *83*)

translations
-since-d ⇌ *CONST since-d*

3.8.2 Axioms

NextUntil

lemma *NextUntilsema*:

assumes ($\sigma \models \bigcirc(f\ \mathcal{U}\ g)$)
shows ($\sigma \models (\bigcirc f)\ \mathcal{U}\ (\bigcirc g)$)
proof −
have *0*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma))\ \ ))$
using *assms zero-enat-def* **by** (*auto simp add*: *next-defs until-d-def ndropn-ndropn*)
have *1*: $0 < nlength\ \sigma$
using *0* **by** *auto*
have *2*: $(\exists k.\ k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ f\ ((\ (ndropn\ (Suc\ j)\ \sigma))\ )))$
using *0* **by** *auto*
obtain *k* **where** *3*: $k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((\ (ndropn\ (Suc\ k)\ \sigma))) \wedge$
$(\forall j<k.\ f\ ((\ (ndropn\ (Suc\ j)\ \sigma))\ ))$
using *2* **by** *auto*
have *4*: $g\ ((\ (ndropn\ (Suc\ k)\ \sigma)))$
using *3* **by** *auto*
have *5*: $k \leq nlength\ \sigma$
using *3 0*
by (*metis diff-le-self dual-order.order-iff-strict enat-ile enat-ord-simps*(*1*) *idiff-enat-enat*
less-le-trans ndropn-nlength not-less)
have *6*: $0 < nlength\ (ndropn\ k\ \sigma)$
using *1 3*
by (*metis gr-zeroI iless-Suc-eq is-NNil-ndropn leD le-numeral-extra*(*3*) *ndropn-0*
ndropn-Suc-conv-ndropn nlength-NCons zero-enat-def)
have *7*: $(\forall j<k.\ 0 < nlength\ (ndropn\ j\ \sigma) \wedge f\ ((\ (ndropn\ (Suc\ j)\ \sigma))))$
using *3 5*
by (*metis enat-ord-simps*(*2*) *is-NNil-ndropn ndropn-0 not-less order.trans zero-le*)
have *71*: $nlength\ \sigma - enat\ k \neq enat\ 0$
using *6 zero-enat-def* **by** *auto*
have *72*: $(\forall j<k.\ nlength\ \sigma - enat\ j \neq enat\ 0 \wedge f\ (ndropn\ (Suc\ j)\ \sigma))$
using *7 zero-enat-def* **by** *auto*
have *8*: $\exists k.\ k \leq nlength\ \sigma \wedge$
$nlength\ \sigma - enat\ k \neq enat\ 0 \wedge$
$g\ ((\ (ndropn\ (Suc\ k)\ \sigma))) \wedge$
$(\forall j<k.\ nlength\ \sigma - enat\ j \neq enat\ 0 \wedge f\ (ndropn\ (Suc\ j)\ \sigma))$
using *3 5 6 71 7 72* **by** *blast*
from *8* **show** *?thesis*
by (*simp add*: *next-defs until-d-def ndropn-ndropn*)
qed

lemma *NextUntilsemb*:
assumes ($\sigma \models (\bigcirc f)\ \mathcal{U}\ (\bigcirc g)$)
shows ($\sigma \models \bigcirc(f\ \mathcal{U}\ g)$)
proof −
have *1*: $\exists k.\ k \leq nlength\ \sigma \wedge 0 < nlength\ (ndropn\ k\ \sigma) \wedge g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ j < nlength\ \sigma \wedge f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *assms*
by (*auto simp add*: *next-defs until-d-def ndropn-ndropn*)
(*metis is-NNil-ndropn le-zero-eq ndropn-0 ndropn-nlength not-less zero-enat-def*)

obtain k **where** *2*: $k \leq nlength\ \sigma \wedge 0 < nlength\ (ndropn\ k\ \sigma) \wedge$
$g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j{<}k.\ j < nlength\ \sigma \wedge f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *1* **by** *auto*
have *3*: $0 < nlength\ \sigma$
using *2* **by** *auto*
have *4*: $k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma)$
by *auto*
(*metis 2 3 add.commute add.right-neutral enat-min iless-Suc-eq ndropn-0*
ndropn-Suc-conv-ndropn ndropn-nlength nlength-NCons zero-enat-def)
have *5*: $g\ (\ (ndropn\ (Suc\ k)\ \sigma))$
using *2* **by** *auto*
have *6*: $(\forall j{<}k.\ j < nlength\ \sigma \wedge f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *2* **by** *blast*
have *7*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j{<}k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma))\ )\ \ )$
using *2 3 4* **by** *blast*
from *7* **show** *?thesis* **by** (*auto simp*: *next-defs until-d-def zero-enat-def ndropn-ndropn*)
qed

lemma *NextUntilsem*:
$\sigma \models \bigcirc(f\ \mathcal{U}\ g) = (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g)$
using *NextUntilsema NextUntilsemb* **using** *unl-lift2* **by** *blast*

lemma *NextUntil*:
$\vdash \bigcirc(f\ \mathcal{U}\ g) = (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g)$
using *NextUntilsem Valid-def* **by** *blast*

UntilNextUntil

lemma *UntilNextUntilsema*:
assumes $0 < nlength\ \sigma \wedge$
$(\exists k.\ 0{<}\ k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma))))$
shows $(\ \sigma) \models \bigcirc\ (\ f\ \mathcal{U}\ g)$
proof −
have *1*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ 0{<}k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma))))$
using *assms* **by** *auto*
have *3*: $(\exists k.\ 0{<}k \wedge k \leq nlength\ \sigma \wedge g\ ((\ (ndropn\ k\ \sigma))) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma)))\ )$
using *1* **by** *auto*
obtain k **where** *4*: $0{<}\ (Suc\ k) \wedge (Suc\ k) \leq nlength\ \sigma \wedge g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j.\ 0 < j \wedge j < (Suc\ k) \longrightarrow f\ (\ (ndropn\ j\ \sigma)))$
using *3* **by** (*metis Suc-pred*)
have *5*: $k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma)$
by (*metis 4 One-nat-def add.commute co.enat.sel(2) eSuc-enat epred-conv-minus le-cases min-absorb1*
ndropn-nlength ntaken-all ntaken-ndropn-swap-nlength ntaken-nlength one-enat-def plus-1-eSuc(2)
plus-1-eq-Suc)
have *6*: $g\ (\ (ndropn\ (Suc\ k)\ \sigma))$
using *4* **by** *auto*

have *7*: $(\forall j{<}k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *4* **by** *blast*
have *8*: $(\exists k.\ k{\leq}nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j{<}k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *4 5* **by** *blast*
have *9*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ k{\leq}nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j{<}k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *1 8* **by** *blast*
from *9* **show** *?thesis* **by** (*auto simp add*: *next-defs until-d-def zero-enat-def ndropn-ndropn*)
qed

lemma *UntilNextUntilsemb*:
assumes $\sigma \models \bigcirc\ (\ f\ \mathcal{U}\ g)$
shows $0 < nlength\ \sigma \wedge$
$(\exists k.\ 0{<}\ k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma))))$
proof −
have *1*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ k{\leq}nlength(ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j{<}k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *assms* **by** (*auto simp add*: *next-defs until-d-def ndropn-ndropn*) (*simp add*: *zero-enat-def*)
have *2*: $(\exists k.\ k{\leq}nlength(ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j{<}k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *1* **by** *auto*
obtain k **where** *3*: $k{\leq}nlength(ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j{<}k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *2* **by** *auto*
have *4*: $0{<}\ (Suc\ k)$
by *simp*
have *5*: $g\ (\ (ndropn\ (Suc\ k)\ \sigma))$
using *3* **by** *auto*
have *6*: $(Suc\ k) \leq nlength\ \sigma$
using *3* **by** *auto*
(*metis 1 dual-order.eq-iff eSuc-enat is-NNil-ndropn le-cases ndropn-0 ndropn-Suc-conv-ndropn*
ndropn-ndropn ndropn-nlength nlength-NCons plus-1-eq-Suc zero-enat-def)
have *7*: $(\forall j.\ 0 < j \wedge j < (Suc\ k) \longrightarrow f\ (\ (ndropn\ j\ \sigma)))$
using *3 less-Suc-eq-0-disj* **by** *auto*
have *8*: $(\exists k.\ 0{<}k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma))))$
using *3 6 7* **by** *blast*
show *?thesis* **using** *1 8* **by** *blast*
qed

lemma *UntilNextUntilsem*:
$(\ \sigma \models \bigcirc\ (\ f\ \mathcal{U}\ g)) =$
$(0 < nlength\ \sigma \wedge$
$(\exists k.\ 0{<}\ k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma)))))$
using *UntilNextUntilsema*[*of* σ g f] *UntilNextUntilsemb*[*of* f g σ] **by** *meson*

lemma *UntilNextUntilsem1*:
$(\sigma \models f\ \mathcal{U}\ g) = (\sigma \models (g \vee (f \wedge \bigcirc(f\ \mathcal{U}\ g))))$
unfolding *UntilNextUntilsem*
proof
assume *a*: $(\sigma \models f\ \mathcal{U}\ g)$
show $(\sigma \models g \vee$

$f \wedge$
($\lambda\sigma.\ \ 0 < nlength\ \sigma \wedge$
$(\exists k.\ 0<k \wedge k \leq nlength\ \sigma \wedge g((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f((ndropn\ j\ \sigma))))$
))
using *a* **by** (*simp add*: *until-d-def*) (*metis enat-0-iff*(*2*) *i0-less ndropn-0 neq0-conv not-le*)
next
next
assume *b*: ($\sigma \models g \vee$
$f \wedge$
($\lambda\sigma.\ 0 < nlength\ \sigma \wedge$
$(\exists k.\ 0<k \wedge k \leq nlength\ \sigma \wedge g((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f((ndropn\ j\ \sigma))))))$
show ($\sigma \models f\ \mathcal{U}\ g$)
using *b* **by** (*simp add*: *until-d-def*) (*metis i0-lb linorder-cases ndropn-0 not-less-zero zero-enat-def*)
qed

lemma *UntilNextUntil*:
$\vdash f\ \mathcal{U}\ g = (g \vee (f \wedge \bigcirc(f\ \mathcal{U}\ g)))$
by (*simp add*: *UntilNextUntilsem1 Valid-def*)

NotUntilFalse

lemma *NotUntilFalse*:
$\vdash \neg\ (f\ \mathcal{U}\ \#False)$
by (*simp add*: *intI until-d-def*)

UntilOrDist

lemma *UntilOrDistsem*:
$\sigma \models f\ \mathcal{U}\ (g \vee h) = (\ f\ \mathcal{U}\ g \vee f\ \mathcal{U}\ h)$
by (*auto simp add*: *until-d-def*)

lemma *UntilOrDist*:
$\vdash f\ \mathcal{U}\ (g \vee h) = (\ f\ \mathcal{U}\ g \vee f\ \mathcal{U}\ h)$
using *UntilOrDistsem Valid-def* **by** *blast*

UntilRightDistOr

lemma *UntilRightDistOr*:
$\vdash f\ \mathcal{U}\ h \vee g\ \mathcal{U}\ h \longrightarrow (f \vee g)\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)

UntilLeftDistAnd

lemma *UntilLeftDistAnd*:
$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow f\ \mathcal{U}\ g \wedge f\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)

UntilAndDist

lemma *UntilAndDistsem*:
$\sigma \models (f \wedge g)\ \mathcal{U}\ h = ((f\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h))$
by (*auto simp add*: *until-d-def*)

(*metis* (*no-types*, *lifting*) *less-le-trans not-less-iff-gr-or-eq order.order-iff-strict*)

lemma *UntilAndDist*:
$\vdash (f \wedge g)\ \mathcal{U}\ h = ((f\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h))$
using *UntilAndDistsem Valid-def* **by** *blast*

untilNotImp

lemma *UntilNotImp*:
$\vdash f\ \mathcal{U}\ g \wedge (\neg\ g)\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h$
by (*simp add*: *Valid-def until-d-def*)
(*metis not-less-iff-gr-or-eq order.strict-trans*)

UntilUntil

lemma *UntilUntilsem*:
$(\sigma \models f\ \mathcal{U}\ g) = (\sigma \models f\ \mathcal{U}\ (f\ \mathcal{U}\ g))$
proof *auto*
show $\sigma \models (f\ \mathcal{U}\ g) \Longrightarrow \sigma \models (f\ \mathcal{U}\ (f\ \mathcal{U}\ g))$
by (*simp add*: *until-d-def*)
(*metis enat-add-sub-same enat-le-plus-same*(*1*) *enat-ord-code*(*4*) *enat-ord-simps*(*4*) *gen-nlength-def ndropn-0 nlength-code not-less-zero*)
show $\sigma \models (f\ \mathcal{U}\ (f\ \mathcal{U}\ g)) \Longrightarrow \sigma \models (f\ \mathcal{U}\ g)$
proof −
assume *a*: $\sigma \models (f\ \mathcal{U}\ (f\ \mathcal{U}\ g))$
show $\sigma \models (f\ \mathcal{U}\ g)$
proof −
have *1*: $\exists k.\ enat\ k \leq nlength\ \sigma \wedge$
$(\exists ka.\ enat\ ka \leq nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall j{<}k.\ f\ (ndropn\ j\ \sigma))$
using *a* **unfolding** *until-d-def* **by** *blast*
obtain *k* **where** *2*:
$enat\ k \leq nlength\ \sigma \wedge$
$(\exists ka.\ enat\ ka \leq nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall j{<}k.\ f\ (ndropn\ j\ \sigma))$
using *1* **by** *auto*
have *3*: $(\exists ka.\ enat\ ka \leq nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma))))$
using *2* **by** *auto*
obtain *ka* **where** *4*:
$enat\ ka \leq nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma)))$
using *3* **by** *auto*
have *41*: $enat\ ka \leq nlength\ \sigma - (enat\ k)$
using *4* **by** *auto*
have *5*: $enat\ (ka{+}k) \leq nlength\ \sigma$
using *2 41* **by** *auto*
(*metis add.commute antisym-conv2 enat.simps*(*3*) *enat-add-sub-same enat-min le-iff-add less-imp-le order-refl plus-enat-simps*(*1*))

have *6*: $g\ (ndropn\ (ka+k)\ \sigma)$
by (*metis 4 add.commute ndropn-ndropn*)
have *7*: $(\forall j{<}(ka+k).\ f\ (ndropn\ j\ \sigma))$
by (*metis 2 4 add-diff-inverse-nat less-diff-conv2 linorder-not-less ndropn-ndropn*)
have *8*: $\exists k.\ k \leq nlength\ \sigma \wedge g\ (ndropn\ k\ \sigma) \wedge (\forall j{<}k.\ f\ (ndropn\ j\ \sigma))$
using *5 6 7* **by** *blast*
show *?thesis* **unfolding** *until-d-def* **by** (*simp add*: *8*)
qed
qed
qed

lemma *UntilUntil*:
$\vdash f\ \mathcal{U}\ g = f\ \mathcal{U}\ (f\ \mathcal{U}\ g)$
using *UntilUntilsem* **by** *fastforce*

UntilRightor

lemma *UntilRightOr*:
$\vdash f\ \mathcal{U}\ (\ g\ \mathcal{U}\ h\ ) \longrightarrow (f \vee g)\ \mathcal{U}\ h$
proof (*auto simp add*: *Valid-def until-d-def ndropn-ndropn*)
fix $w :: 'a\ nellist$
fix k
fix ka
assume *a0*: $enat\ k \leq nlength\ w$
assume *a1*: $\forall j{<}k.\ f\ (ndropn\ j\ w)$
assume *a2*: $enat\ ka \leq nlength\ w - enat\ k$
assume *a3*: $h\ (ndropn\ (k + ka)\ w)$
assume *a4*: $\forall j{<}ka.\ g\ (ndropn\ (k + j)\ w)$
show $\exists k.\ enat\ k \leq nlength\ w \wedge h\ (ndropn\ k\ w) \wedge (\forall j{<}k.\ f\ (ndropn\ j\ w) \vee g\ (ndropn\ j\ w))$
proof −
have *1*: $ka+k \leq nlength\ w$
by (*metis a0 a2 add.commute dual-order.order-iff-strict enat.simps*(*3*) *enat-add-sub-same enat-less-enat-plusI2 less-eqE plus-enat-simps*(*1*))
have *2*: $h\ (ndropn\ (ka+k)\ w)$
using *a3* **by** (*simp add*: *add.commute*)
have *3*: $(\forall j{<}(ka+k).\ f\ (ndropn\ j\ w) \vee g\ (ndropn\ j\ w))$
by (*metis a1 a4 add-diff-inverse-nat less-diff-conv2 not-less*)
show *?thesis* **using** *1 2 3* **by** *blast*
qed
qed

UntilRightAnd

lemma *UntilRightAndsem*:
assumes $(\sigma \models f\ \mathcal{U}\ (g \wedge h))$
shows $(\sigma \models (f\ \mathcal{U}\ g)\ \mathcal{U}\ h)$
proof −
have *1*: $\exists k.\ k{\leq}nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge h\ ((ndropn\ k\ \sigma)) \wedge (\forall j{<}k.\ f\ ((ndropn\ j\ \sigma)))$
using *assms* **by** (*simp add*: *until-d-def*)
obtain k **where** *2*: $k{\leq}nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge h\ ((ndropn\ k\ \sigma)) \wedge (\forall j{<}k.\ f\ ((ndropn\ j\ \sigma)))$
using *1* **by** *auto*

have *3*: $h\ (\ (ndropn\ k\ \sigma))$
using *2* **by** *auto*
have *4*: $k{\leq}nlength\ \sigma$
using *2* **by** *auto*
have *5*: $(\forall j{<}k.$
$\exists ka.\ ka{\leq}nlength\ (ndropn\ j\ \sigma) \wedge g\ ((ndropn\ (ka + j)\ \sigma)) \wedge (\forall ja{<}ka.\ f\ ((ndropn\ (ja + j)\ \sigma))))$
proof *auto*
fix *j*
assume *a0*: $j < k$
show $\exists ka.\ enat\ ka \leq nlength\ \sigma - enat\ j \wedge g\ (ndropn\ (ka + j)\ \sigma) \wedge (\forall ja{<}ka.\ f\ (ndropn\ (ja + j)\ \sigma))$
proof −
have *51*: $k{-}j \leq nlength\ (ndropn\ j\ \sigma)$
using *4 a0*
by (*metis enat-minus-mono1 idiff-enat-enat ndropn-nlength*)
have *52*: $g\ (\ (ndropn\ ((k{-}j) + j)\ \sigma))$
by (*simp add*: *2 a0 less-imp-le-nat*)
have *53*: $(\forall ja{<}(k{-}j).\ f\ (\ (ndropn\ (ja + j)\ \sigma)))$
using *2 less-diff-conv* **by** *blast*
show *?thesis*
using *51 52 53* **by** *auto*
qed
qed
have *6*: $\exists k.\ k{\leq}nlength\ \sigma \wedge$
$h\ (\ (ndropn\ k\ \sigma)) \wedge$
$(\forall j{<}k.\ \exists k{\leq}nlength\ (ndropn\ j\ \sigma).\ g\ ((ndropn\ (k + j)\ \sigma)) \wedge (\forall ja{<}k.\ f\ ((ndropn\ (ja + j)\ \sigma))))$
using *2 5* **by** *blast*
from *6* **show** *?thesis* **by** (*simp add*: *until-d-def ndropn-ndropn add.commute*)
qed

lemma *UntilRightAnd*:
$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow (f\ \mathcal{U}\ g)\ \mathcal{U}\ h$
using *UntilRightAndsem Valid-def* **by** *auto*

DiamondEqvTrueUntil

lemma *DiamondEqvTrueUntil*:
$\vdash \Diamond f = \#True\ \mathcal{U}\ f$
by (*simp add*: *Valid-def sometimes-defs until-d-def*)

TrueUntilImpNotUntil

lemma *nellist-ndropn-first-upto*:
assumes $(\exists i{\leq}\ k.\ f\ (\ (ndropn\ i\ xs)))$
shows $(\exists i{\leq}\ k.\ f\ (\ (ndropn\ i\ xs)) \wedge (\forall j{<}i\ .\ \neg\ (f\ (\ (ndropn\ j\ xs)))))$
using *assms*
proof (*induct k arbitrary*: *xs*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*

by (*metis le-Suc-eq less-Suc-eq-le*)
qed

lemma *nellist-ndropn-first*:
assumes ($\exists\, i{\leq}$*nlength xs*. *f* ((*ndropn i xs*)))
shows ($\exists\, i{\leq}$*nlength xs*. *f* ((*ndropn i xs*)) $\wedge$ ($\forall\, j$. *j*<*i* $\longrightarrow$ $\neg$ (*f* ((*ndropn j xs*)))))
proof (*cases nfinite xs*)
case *True*
then show *?thesis* **using** *assms nellist-ndropn-first-upto*[*of - f xs*] *nfinite-nlength-enat*[*of xs*]
by *force*
next
case *False*
then show *?thesis* **using** *assms nellist-ndropn-first-upto*[*of - f xs*]
proof −
assume *a1*: $\bigwedge k.\ \exists\, i{\leq}k.\ f$ (*ndropn i xs*) $\Longrightarrow$ $\exists\, i{\leq}k.\ f$ (*ndropn i xs*) $\wedge$ ($\forall\, j{<}i.\ \neg\ f$ (*ndropn j xs*))
obtain *nn* :: *nat* **where**
f2: *f* (*ndropn nn xs*) $\wedge$ *enat nn* $\leq$ *nlength xs*
using *assms* **by** *blast*
then have $\forall\, e.\ \neg\ e \leq$ *enat nn* $\vee$ *e* $\leq$ *nlength xs*
by *force*
then show *?thesis*
using *f2 a1* **by** (*meson enat-ord-simps*(*1*) *less-imp-le-nat*)
qed
qed

lemma *NotSuffixFirstfinite*:
assumes ($\exists\, n{\leq}$*nlength xs*. $\neg$ *f* ((*ndropn n xs*)))
shows ($\exists\, n \leq$ *nlength xs*. $\neg$ *f* ((*ndropn n xs*)) $\wedge$ ($\forall\, k.\ k{<}n \longrightarrow$ *f* ((*ndropn k xs*))))
using *assms nellist-ndropn-first*[*of xs LIFT*($\neg$ *f*)] **by** *auto*

lemma *TrueUntilImpNotUntilsem*:
assumes $\sigma \models \#\mathit{True}\ \mathcal{U}\ g$
shows $\sigma \models (\neg g)\ \mathcal{U}\ g$
using *assms*
by (*simp add*: *until-d-def nellist-ndropn-first*)

lemma *TrueUntilImpNotUntil*:
$\vdash \#\mathit{True}\ \mathcal{U}\ g \longrightarrow (\neg g)\ \mathcal{U}\ g$
by (*simp add*: *intI nellist-ndropn-first until-d-def*)

WaitNotDistUntil

lemma *WaitNotDistUntilsem1*:
assumes ($\sigma \models \neg(f\ \mathcal{W}\ g)$)
shows ($\sigma \models ((\neg\ g)\ \mathcal{U}\ ((\neg\ f) \wedge (\neg g))))$
proof −
have *1*: ($\forall\, k.\ g$ ((*ndropn k* σ)) $\longrightarrow$ *k* $\leq$ *nlength* $\sigma \longrightarrow$ ($\exists\, j{<}k.\ \neg\ f$ ((*ndropn j* σ)))) $\wedge$
($\exists\, n{\leq}$*nlength* σ. $\neg\ f$ ((*ndropn n* σ)))
using *assms* **by** (*simp add*: *wait-d-def until-d-def always-defs*)
have *2*: ($\forall\, k.\ k \leq$ *nlength* $\sigma \longrightarrow \neg\ g$ ((*ndropn k* σ)) $\vee$ ($\exists\, j{<}k.\ \neg\ f$ ((*ndropn j* σ))))

using *1* **by** *auto*
have *3*: $(\exists\, n{\leq}nlength\ \sigma.\ \neg\ f\ (\ (ndropn\ n\ \sigma)))$
using *1* **by** *auto*
obtain *n* **where** *4*: $n{\leq}\ nlength\ \sigma \wedge \neg\ f\ (\ (ndropn\ n\ \sigma)) \wedge$
$(\forall\ k < n\ .\ f\ (\ (ndropn\ k\ \sigma)))$
using *3* **using** *NotSuffixFirstfinite* **by** *blast*
have *16*: $n \leq\ nlength\ \sigma$
by (*simp add*: *4*)
have *17*: $\neg\ g\ (\ (ndropn\ n\ \sigma))$
using *1 4* **by** *blast*
have *18*: $(\forall\, j{<}n.\ \neg\ g\ (\ (ndropn\ j\ \sigma)))$
by (*metis 2 4 dual-order.strict-iff-order dual-order.strict-trans1 enat-ord-simps*(*2*))
have *19*: $\exists\, k{\leq}nlength\ \sigma.\ \neg\ f\ (\ (ndropn\ k\ \sigma)) \wedge \neg\ g\ (\ (ndropn\ k\ \sigma)) \wedge (\forall\, j{<}k.\ \neg\ g\ (\ (ndropn\ j\ \sigma)))$
using *16 17 18 4* **by** *blast*
have *20*: $(\ \sigma \models ((\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg g)))$
using *19* **by** (*simp add*: *until-d-def*)
show *?thesis* **using** *20* **by** *auto*
qed

lemma *WaitNotDistUntilsem2*:
assumes $(\sigma \models ((\neg\ g)\ \mathcal{U}\ ((\neg\ f) \wedge (\neg g))))$
shows $(\ \sigma \models \neg(f\ \mathcal{W}\ g))$
using *assms not-less-iff-gr-or-eq* **by** (*auto simp add*: *always-defs wait-d-def until-d-def*)

lemma *WaitNotDistUntilsem*:
$(\sigma \models (\neg(f\ \mathcal{W}\ g)) \ =\ ((\neg\ g)\ \mathcal{U}\ ((\neg\ f) \wedge (\neg g))))$
using *WaitNotDistUntilsem1 WaitNotDistUntilsem2*
unl-lift2 **by** *blast*

lemma *WaitNotDistUntil*:
$\vdash (\neg(f\ \mathcal{W}\ g)) = ((\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg g))$
using *WaitNotDistUntilsem* *Valid-def* **by** (*metis*)

UntilInduction

lemma *LFPUntilsem1*:
assumes $\forall\, n{\leq}nlength\ \sigma.$
$(g\ (\ (ndropn\ n\ \sigma)) \longrightarrow h\ (\ (ndropn\ n\ \sigma))) \wedge$
$(f\ (\ (ndropn\ n\ \sigma)) \wedge n < nlength\ \sigma \wedge h\ (\ (ndropn\ (Suc\ n)\ \sigma)) \longrightarrow$
$h\ (\ (ndropn\ n\ \sigma)))$
$k \leq\ nlength\ \sigma$
$g\ (\ (ndropn\ k\ \sigma))$
$\forall\, j{<}k.\ f\ (\ (ndropn\ j\ \sigma))$
shows $h\ (\ \sigma)$
using *assms*
proof (*induct k arbitrary*: σ)
case *0*
then show *?case* **by** *auto*
next
case (*Suc k*)

then show *?case*
proof −
have *1*: $g\ (ndropn\ (Suc\ k)\ \sigma) \longrightarrow h\ \ (ndropn\ (Suc\ k\ )\ \sigma)$
using *Suc.prems(1) Suc.prems(2)* **by** *blast*
have *2*: $(f\ (\ (ndropn\ k\ \sigma)) \wedge k < nlength\ \sigma \wedge h\ (\ (ndropn\ (Suc\ k)\ \sigma)) \longrightarrow$
$h\ (\ (ndropn\ k\ \sigma)))$
by (*simp add*: *Suc.prems(1) less-le-not-le*)
have *3*: $k < nlength\ \sigma$
using *Suc.prems(2) Suc-ile-eq* **by** *auto*
have *4*: $f\ (\ (ndropn\ k\ \sigma))$
by (*simp add*: *Suc.prems(4)*)
have *5*: $h\ (\ (ndropn\ k\ \sigma))$
using *1 2 3 4 Suc.prems(3)* **by** *blast*
have *6*: $h\ (\ (ndropn\ 0\ \sigma))$
using *zero-induct*[*of* $\lambda k\ .\ h\ (ndropn\ k\ \ \sigma)\ k$]
3 5 Suc.prems nellist-ndropn-first[*of* $\sigma\ h$]
by (*metis Suc-ile-eq less-Suc-eq-0-disj less-imp-le not-less-eq*)
show *?thesis*
using *6* **by** *auto*
qed
qed

lemma *LFPUntilsem*:
$\sigma \models \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h)$
using *LFPUntilsem1*[*of* $\sigma\ g\ h\ f$]
by (*auto simp add*: *always-defs next-defs until-d-def ndropn-ndropn*)
(*metis canonically-ordered-monoid-add-class.lessE enat.simps(3) enat-add-sub-same zero-enat-def*)

lemma *LFPUntil*:
$\vdash \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h)$
using *LFPUntilsem Valid-def*
by (*metis*)

lemma *UntilInduction-a*:
$\vdash \Box(f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow (f \longrightarrow \Box\ g \vee g\ \mathcal{U}\ h)$
proof −
have *1*: $\vdash (\Box\ g \vee g\ \mathcal{U}\ h) = g\ \mathcal{W}\ h$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash (f \longrightarrow \Box\ g \vee g\ \mathcal{U}\ h) = (\ (\neg\ h)\ \mathcal{U}\ (\neg\ g \wedge \neg h) \longrightarrow \neg\ f)$
using *1 WaitNotDistUntil* **by** *fastforce*
have *3*: $\vdash \Box(\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f))) \longrightarrow \neg f) \longrightarrow (\ (\neg\ h)\ \mathcal{U}\ (\neg\ g \wedge \neg h) \longrightarrow \neg\ f)$
using *LFPUntil* **by** *blast*
have *4*: $\vdash (f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow (\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f))) \longrightarrow \neg f)$
using *NextImpNotNextNot*[*of f*] **by** *auto*
have *5*: $\vdash \Box(f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow \Box(\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f))) \longrightarrow \neg f)$
using *4* **by** (*rule ImpBoxRule*)
show *?thesis*
using *2 3 5* **by** *fastforce*
qed

lemma *UntilInduction-b*:
$\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow (f \longrightarrow \Box\, f \vee f\ \mathcal{U}\ g)$
proof −
have *1*: $\vdash (\Box\, f \vee f\ \mathcal{U}\ g) = f\ \mathcal{W}\ g$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash (f \longrightarrow \Box\, f \vee f\ \mathcal{U}\ g) = (\ (\neg\, g)\ \mathcal{U}\ (\neg\, f \wedge \neg\, g) \longrightarrow \neg\, f)$
using *1 WaitNotDistUntil* **by** *fastforce*
have *3*: $\vdash \Box\ (\ ((\neg\, f \wedge \neg\, g) \vee (\neg g \wedge \bigcirc\ (\neg\, f))) \longrightarrow \neg\, f) \longrightarrow (\ (\neg\, g)\ \mathcal{U}\ (\neg\, f \wedge \neg\, g) \longrightarrow \neg\, f)$
using *LFPUntil* **by** *blast*
have *4*: $\vdash (f \longrightarrow (\bigcirc f) \vee g) \longrightarrow\ \ (\ ((\neg\, f \wedge \neg\, g) \vee (\neg g \wedge \bigcirc\ (\neg\, f))) \longrightarrow \neg\, f)$
using *NextImpNotNextNot*[*of f*] **by** *auto*
have *5*: $\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow\ \ \Box(\ ((\neg\, f \wedge \neg\, g) \vee (\neg g \wedge \bigcirc\ (\neg\, f))) \longrightarrow \neg\, f)$
using *4 BoxImpBoxRule* **by** *blast*
show *?thesis*
using *2 3 5* **by** *fastforce*
qed

3.8.3 Theorems

lemma *NextFalseSUntil*:
$\vdash \bigcirc\ g = \#False\ \mathcal{U}^s\ g$
proof −
have *1*: $\vdash \#False\ \mathcal{U}\ g = g$
using *UntilNextUntil*[*of LIFT*(*#False*) *g*] **by** *auto*
show *?thesis* **unfolding** *suntil-d-def* **using** *1 inteq-reflection* **by** *force*
qed

lemma *WNextUntil*:
$\vdash wnext(f\ \mathcal{U}\ g) = (empty \vee (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g))$
by (*meson NextUntil Prop06 WnextEqvEmptyOrNext*)

lemma *UntilRelease*:
$\vdash f\ \mathcal{R}\ g = (\neg\ (\ (\neg\, f)\ \mathcal{U}\ (\neg\, g)))$
by (*simp add*: *release-d-def*)

lemma *SReleaseWait*:
$\vdash f\ \mathcal{M}\ g = (\neg\ (\neg\, f)\ \mathcal{W}\ (\neg\, g))$
by (*simp add*: *srelease-d-def*)

lemma *ReleaseUntil*:
$\vdash f\ \mathcal{U}\ g = (\neg\ (\ (\neg\, f)\ \mathcal{R}\ (\neg\, g)))$
by (*simp add*: *release-d-def*)

lemma *WaitSRelease*:
$\vdash f\ \mathcal{W}\ g = (\neg\ (\neg\, f)\ \mathcal{M}\ (\neg\, g))$
by (*simp add*: *srelease-d-def*)

lemma *NotUntilRelease*:
$\vdash \neg(f\ \mathcal{U}\ g) = (\neg\, f)\ \mathcal{R}\ (\neg\, g)$
by (*simp add*: *ReleaseUntil*)

lemma *NotWaitSRelease*:
$\vdash \neg(f\ \mathcal{W}\ g) = (\neg\ f)\ \mathcal{M}\ (\neg\ g)$
by (*simp add*: *WaitSRelease*)

lemma *NotReleaseUntil*:
$\vdash \neg(f\ \mathcal{R}\ g) = (\neg\ f)\ \mathcal{U}\ (\neg\ g)$
by (*simp add*: *UntilRelease*)

lemma *NotSReleaseWait*:
$\vdash \neg(f\ \mathcal{M}\ g) = (\neg\ f)\ \mathcal{W}\ (\neg\ g)$
by (*simp add*: *SReleaseWait*)

lemma *BoxEqvFalseRelease*:
$\vdash \Box\ f = \#False\ \mathcal{R}\ f$
unfolding *release-d-def*
by (*metis DiamondEqvTrueUntil Prop11 always-d-def int-simps*(*3*) *inteq-reflection lift-imp-neg*)

lemma *UntilTrue*:
$\vdash f\ \mathcal{U}\ \#True$
using *UntilNextUntil* **by** *fastforce*

lemma *UntilIdempotent*:
$\vdash f\ \mathcal{U}\ f = f$
using *UntilNextUntil*[*of f f*] **by** *auto*

lemma *UntilImpUntil*:
assumes $\vdash f0 \longrightarrow f1$
$\vdash g0 \longrightarrow g1$
shows $\vdash f0\ \mathcal{U}\ g0 \longrightarrow f1\ \mathcal{U}\ g1$
using *assms*
by (*metis Prop10 Prop12 UntilAndDist UntilLeftDistAnd int-eq*)

lemma *UntilEqvUntil*:
assumes $\vdash f0 = f1$
$\vdash g0 = g1$
shows $\vdash f0\ \mathcal{U}\ g0 = f1\ \mathcal{U}\ g1$
proof −
have *1*: $\vdash f0 \longrightarrow f1$
using *assms* **by** *auto*
have *2*: $\vdash g0 \longrightarrow g1$
using *assms* **by** *auto*
have *3*: $\vdash f0\ \mathcal{U}\ g0 \longrightarrow f1\ \mathcal{U}\ g1$
using *1 2 UntilImpUntil*[*of f0 f1 g0 g1*] **by** *auto*
have *4*: $\vdash f1 \longrightarrow f0$
using *assms* **by** *auto*
have *5*: $\vdash g1 \longrightarrow g0$
using *assms* **by** *auto*
have *6*: $\vdash f1\ \mathcal{U}\ g1 \longrightarrow f0\ \mathcal{U}\ g0$
using *4 5 UntilImpUntil*[*of f1 f0 g1 g0*] **by** *auto*

from *3 6* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilRightDistImp*:
$\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
proof −
have *1*: $\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h) =$
$((f \longrightarrow g)\ \mathcal{U}\ h \wedge f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
by *auto*
have *2*: $\vdash ((f \longrightarrow g)\ \mathcal{U}\ h \wedge f\ \mathcal{U}\ h) = ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h$
by (*simp add*: *UntilAndDist int-iffD1 int-iffD2 int-iffI*)
have *3*: $\vdash ((f \longrightarrow g) \wedge f) = (f \wedge g)$
by *auto*
have *4*: $\vdash h = h$
by *auto*
have *5*: $\vdash ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h = (f \wedge g)\ \mathcal{U}\ h$
using *3 4* **using** *UntilEqvUntil* **by** *blast*
have *6*: $\vdash (f \wedge g)\ \mathcal{U}\ h = (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)$
by (*simp add*: *UntilAndDist*)
show *?thesis*
using *2 5 6* **by** *fastforce*
qed

lemma *FalseUntil*:
$\vdash \#False\ \mathcal{U}\ g = g$
by (*metis Prop10 Prop12 TrueW UntilNextUntil int-simps*(*14*) *int-simps*(*21*) *int-simps*(*25*) *int-simps*(*3*)
inteq-reflection)

lemma *UntilExclMid*:
$\vdash f\ \mathcal{U}\ g \vee f\ \mathcal{U}\ (\neg\ g)$
using *UntilOrDist UntilTrue* **by** *fastforce*

lemma *NotUntilImp*:
$\vdash (\neg\ f)\ \mathcal{U}\ (g\ \mathcal{U}\ h) \wedge f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h$
proof −
have *1*: $\vdash (\neg\ f)\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow (\neg\ f \vee g)\ \mathcal{U}\ h$
by (*simp add*: *UntilRightOr*)
have *2*: $\vdash (\neg\ f \vee g) = (f \longrightarrow g)$
by *auto*
have *3*: $\vdash h = h$
by *auto*
have *4*: $\vdash\ (\neg\ f \vee g)\ \mathcal{U}\ h = (f \longrightarrow g)\ \mathcal{U}\ h$
by (*simp add*: *2 UntilEqvUntil*)
have *5*: $\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
by (*simp add*: *UntilRightDistImp*)
have *6*: $\vdash (\neg\ f)\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
using *1 4 5* **by** *fastforce*
from *6* **show** *?thesis* **by** *auto*
qed

lemma *UntilNotImpa*:
$\vdash f\ \mathcal{U}\ ((\neg\ g)\ \mathcal{U}\ h) \wedge g\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h$
proof −
have *1*: $\vdash f\ \mathcal{U}\ ((\neg\ g)\ \mathcal{U}\ h) \longrightarrow (f \vee (\neg\ g))\ \mathcal{U}\ h$
by (*simp add*: *UntilRightOr*)
have *2*: $\vdash (f \vee (\neg\ g)) = (g \longrightarrow f)$
by *auto*
have *3*: $\vdash h = h$
by *auto*
have *4*: $\vdash (f \vee (\neg\ g))\ \mathcal{U}\ h = (g \longrightarrow f)\ \mathcal{U}\ h$
by (*simp add*: *2 UntilEqvUntil*)
have *5*: $\vdash (g \longrightarrow f)\ \mathcal{U}\ h \longrightarrow (\ g\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h)$
by (*simp add*: *UntilRightDistImp*)
have *6*: $\vdash\ f\ \mathcal{U}\ ((\neg\ g)\ \mathcal{U}\ h) \longrightarrow (\ g\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h)$
using *1 4 5* **by** *fastforce*
from *6* **show** *?thesis* **by** *auto*
qed

lemma *UntilNotUntilImp*:
$\vdash f\ \mathcal{U}\ g \wedge (\neg\ g)\ \mathcal{U}\ f \longrightarrow f$
proof −
have *1*: $\vdash f\ \mathcal{U}\ g \wedge (\neg\ g)\ \mathcal{U}\ f \longrightarrow f\ \mathcal{U}\ f$
using *UntilNotImp* **by** *auto*
have *2*: $\vdash\ f\ \mathcal{U}\ f = f$
using *UntilIdempotent* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *AndNotUntilImp*:
$\vdash f \wedge (\neg\ f)\ \mathcal{U}\ g \longrightarrow g$
proof −
have *1*: $\vdash f = f\ \mathcal{U}\ f$
by (*simp add*: *UntilIdempotent int-iffD1 int-iffD2 int-iffI*)
have *2*: $\vdash g = \ \#False\ \mathcal{U}\ g$
by (*meson FalseUntil Prop11*)
have *3*: $\vdash f\ \mathcal{U}\ f \wedge (\neg\ f)\ \mathcal{U}\ g \longrightarrow \#False\ \mathcal{U}\ g$
by (*metis 1 FalseUntil UntilNotImp inteq-reflection*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilImpOr*:
$\vdash f\ \mathcal{U}\ g \longrightarrow f \vee g$
proof −
have $\vdash f \wedge \bigcirc(f\ \mathcal{U}\ g) \longrightarrow f \vee g$
by *force*
then show *?thesis*
using *UntilNextUntil*[*of f g*] **by** *auto*
qed

lemma *UntilIntro*:

$\vdash g \longrightarrow f\ \mathcal{U}\ g$
proof −
have *1*: $\vdash g = \#False\ \mathcal{U}\ g$
by (*meson FalseUntil Prop11*)
have *2*: $\vdash \#False \longrightarrow f$
by *auto*
have *3*: $\vdash g \longrightarrow g$
by *auto*
have *4*: $\vdash \#False\ \mathcal{U}\ g \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *UntilImpUntil*)
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *OrImpUntil*:
$\vdash f \wedge g \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *Prop01 Prop05 UntilIntro*)

lemma *UntilAbsorp-a*:
$\vdash (f \vee f\ \mathcal{U}\ g) = (f \vee g)$
proof −
have *1*: $\vdash (f \vee f\ \mathcal{U}\ g) \longrightarrow f \vee g$
using *UntilImpOr* **by** *fastforce*
have *2*: $\vdash f \vee g \longrightarrow (f \vee f\ \mathcal{U}\ g)$
using *UntilIntro* **by** *fastforce*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilAbsorp-b*:
$\vdash (f\ \mathcal{U}\ g \vee g) = f\ \mathcal{U}\ g$
using *UntilNextUntil* **by** *fastforce*

lemma *UntilAbsorp-c*:
$\vdash (f\ \mathcal{U}\ g \wedge g) = g$
using *UntilIntro* **by** *fastforce*

lemma *UntilAbsorp-d*:
$\vdash (f\ \mathcal{U}\ g \vee (f \wedge g)) = f\ \mathcal{U}\ g$
using *UntilNextUntil* **by** *fastforce*

lemma *UntilAbsorp-e*:
$\vdash (f\ \mathcal{U}\ g \wedge (f \vee g)) = f\ \mathcal{U}\ g$
by (*meson Prop10 Prop11 UntilImpOr*)

lemma *LeftUntilAbsorp*:
$\vdash f\ \mathcal{U}\ (f\ \mathcal{U}\ g) = f\ \mathcal{U}\ g$
by (*meson Prop11 UntilUntil*)

lemma *RightUntilAbsorp*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*metis Prop11 UntilAbsorp-b UntilAbsorp-c UntilImpOr UntilRightAnd UntilUntil inteq-reflection*)

lemma *UntilAbsorpAndDiamond*:
$\vdash (f\ \mathcal{U}\ g \wedge \Diamond\ g) = f\ \mathcal{U}\ g$
by (*metis DiamondEqvTrueUntil Prop11 Prop12 UntilIdempotent UntilImpUntil int-simps*(*12*) *inteq-reflection*)

lemma *UntilAbsorpOrDiamond*:
$\vdash (f\ \mathcal{U}\ g \vee \Diamond\ g) = \Diamond\ g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *UntilAbsorpDiamond*:
$\vdash f\ \mathcal{U}\ (\Diamond\ g) = \Diamond\ g$
using *DiamondDiamondEqvDiamond UntilAbsorpOrDiamond UntilAbsorp-b* **by** *fastforce*

lemma *UntilImpDiamond*:
$\vdash f\ \mathcal{U}\ g \longrightarrow \Diamond\ g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *AlwaysImpNotUntilNot*:
$\vdash \Box\ f \longrightarrow \neg(g\ \mathcal{U}\ (\neg\ f))$
by (*simp add*: *UntilImpDiamond always-d-def*)

lemma *UntilAlwaysAndDist*:
$\vdash \Box\ f \wedge g\ \mathcal{U}\ h \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h)$
proof −
have *1*: $\vdash \Box(h \vee g \wedge \bigcirc\ ((f \wedge g)\ \mathcal{U}\ (f \wedge h)) \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h)) \longrightarrow$
$g\ \mathcal{U}\ h \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h)$
using *LFPUntil* **by** *blast*
have *2*: $\vdash\ f \longrightarrow (h \vee g \wedge \bigcirc\ ((f \wedge g)\ \mathcal{U}\ (f \wedge h)) \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h))$
using *UntilNextUntil*[*of LIFT*($f \wedge g$) *LIFT*($f \wedge h$)] **by** *auto*
have *3*: $\vdash\ \Box\ f \longrightarrow \Box(h \vee g \wedge \bigcirc\ ((f \wedge g)\ \mathcal{U}\ (f \wedge h)) \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h))$
using *2 BoxImpBoxRule* **by** *blast*
have *4*: $\vdash \Box\ f\ \longrightarrow (g\ \mathcal{U}\ h \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h))$
using *1 3 lift-imp-trans* **by** *blast*
show *?thesis* **using** *4* **by** *fastforce*
qed

lemma *UntilAndImp*:
$\vdash \Box\ f \wedge \Diamond\ g \longrightarrow f\ \mathcal{U}\ g$
proof −
have *1*: $\vdash \Diamond\ g = \#True\ \mathcal{U}\ g$
by (*simp add*: *DiamondEqvTrueUntil*)
have *2*: $\vdash \Box\ f \wedge \#True\ \mathcal{U}\ g \longrightarrow (f \wedge \#True)\ \mathcal{U}\ (f \wedge g)$
using *UntilAlwaysAndDist* **by** *blast*
have *3*: $\vdash (f \wedge \#True)\ \mathcal{U}\ (f \wedge g) = f\ \mathcal{U}\ (f \wedge g)$
by *simp*
have *4*: $\vdash f\ \mathcal{U}\ (f \wedge g) \longrightarrow (f\ \mathcal{U}\ f)\ \mathcal{U}\ g$
by (*simp add*: *UntilRightAnd*)
have *5*: $\vdash (f\ \mathcal{U}\ f) = f$
by (*simp add*: *UntilIdempotent*)
have *6*: $\vdash (f\ \mathcal{U}\ f)\ \mathcal{U}\ g\ = f\ \mathcal{U}\ g$

by (*simp add*: *5 UntilEqvUntil*)
show *?thesis*
by (*metis 1 2 3 4 5 inteq-reflection lift-imp-trans*)
qed

lemma *UntilRightMono*:
$\vdash \Box(f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \longrightarrow h\ \mathcal{U}\ g)$
proof −
have *1*: $\vdash \Box(f \longrightarrow g) \wedge h\ \mathcal{U}\ f \longrightarrow (\ (f \longrightarrow g) \wedge h)\ \mathcal{U}\ ((f \longrightarrow g) \wedge f)$
using *UntilAlwaysAndDist* **by** *blast*
have *2*: $\vdash ((f \longrightarrow g) \wedge h)\ \mathcal{U}\ ((f \longrightarrow g) \wedge f) \longrightarrow h\ \mathcal{U}\ ((f \longrightarrow g) \wedge f)$
by (*meson Prop12 UntilImpUntil int-iffD2 lift-and-com*)
have *3*: $\vdash ((f \longrightarrow g) \wedge f) \longrightarrow g$
by *auto*
have *4*: $\vdash\ h\ \mathcal{U}\ ((f \longrightarrow g) \wedge f) \longrightarrow h\ \mathcal{U}\ g$
by (*simp add*: *3 UntilImpUntil*)
show *?thesis*
by (*meson 1 2 4 Prop09 lift-imp-trans*)
qed

lemma *UntilLeftMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
proof −
have *1*: $\vdash \Box\ (f \longrightarrow g) \wedge f\ \mathcal{U}\ h \longrightarrow ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ ((f \longrightarrow g) \wedge h)$
by (*simp add*: *UntilAlwaysAndDist*)
have *2*: $\vdash ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ ((f \longrightarrow g) \wedge h) \longrightarrow ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h$
by (*meson Prop12 UntilLeftDistAnd*)
have *3*:$\vdash ((f \longrightarrow g) \wedge f) \longrightarrow g$
by *auto*
have *4*: $\vdash ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h$
by (*simp add*: *3 UntilImpUntil*)
show *?thesis*
by (*meson 1 2 4 Prop09 lift-imp-trans*)
qed

lemma *UntilCatRule*:
$\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow (f \longrightarrow (g\ \mathcal{U}\ i))$
proof −
have *1*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow \Box\ (f \longrightarrow g\ \mathcal{U}\ h)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *2*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow \Box\ (h \longrightarrow g\ \mathcal{U}\ i)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *3*: $\vdash \Box\ (h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow \Box\ (\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ (g\ \mathcal{U}\ i))$
by (*metis BoxEqvBoxBox BoxImpBoxRule UntilRightMono inteq-reflection*)
have *4*: $\vdash \Box\ (\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ (g\ \mathcal{U}\ i)) \longrightarrow \Box(\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
by (*metis BoxEqvBoxBox UntilUntil int-iffD1 inteq-reflection*)
have *5*: $\vdash\ \Box\ (f \longrightarrow g\ \mathcal{U}\ h) \longrightarrow (f \longrightarrow g\ \mathcal{U}\ h)$
by (*simp add*: *BoxElim*)
have *6*: $\vdash \Box(\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
by (*simp add*: *BoxElim*)

have *7*: $\vdash (f \longrightarrow g\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow (f \longrightarrow g\ \mathcal{U}\ i)$
by *auto*
have *8*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow$
$(f \longrightarrow g\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
using *1 2 3 4 5 6* **by** *fastforce*
from *7 8* **show** *?thesis* **by** *auto*
qed

lemma *UntilStrengthen*:
$\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
proof −
have *11*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow \Box(f \longrightarrow h)$
by (*meson BoxImpBoxRule Prop12 int-iffD2 lift-and-com*)
have *1*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g)$
using *11 UntilLeftMono*[*of f h g*] **by** *fastforce*
have *21*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow \Box(g \longrightarrow i)$
by (*simp add*: *BoxImpBoxRule Prop01 Prop05*)
have *2*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
using *21 UntilRightMono*[*of g i h*] **by** *fastforce*
have *3*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g) \wedge (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
using *1 2* **by** *fastforce*
have *4*: $\vdash (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g) \wedge (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
by *auto*
from *3 4* **show** *?thesis* **by** *auto*
qed

lemma *UntilInduction*:
$\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc f) \longrightarrow (f \longrightarrow \neg(h\ \mathcal{U}\ g))$
proof −
have *1*: $\vdash \Box\ (\neg\ g) \longrightarrow \neg(h\ \mathcal{U}\ g)$
by (*simp add*: *UntilImpDiamond always-d-def*)
have *15*: $\vdash (f \longrightarrow \neg\ g \wedge \bigcirc f) \longrightarrow (g \vee\ \bigcirc (\neg\ f) \longrightarrow \neg\ f)$
using *NextImpNotNextNot*[*of f*] **by** *fastforce*
have *16*: $\vdash (f \longrightarrow \neg\ g \wedge \bigcirc f) \longrightarrow (g \vee\ \#True \wedge \bigcirc (\neg\ f) \longrightarrow \neg\ f)$
using *15* **by** *auto*
have *2*: $\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc f) \longrightarrow \Box(g \vee \#True \wedge \bigcirc (\neg\ f) \longrightarrow \neg\ f)$
using *16 BoxImpBoxRule* **by** *blast*
have *3*: $\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc f) \longrightarrow (\#True\ \mathcal{U}\ g \longrightarrow \neg\ f)$
using *2 LFPUntil*[*of g LIFT*(*#True*) *LIFT*($\neg\ f$)]
by *fastforce*
have *4*: $\vdash (\#True\ \mathcal{U}\ g \longrightarrow \neg\ f) \longrightarrow (f \longrightarrow \neg(\#True\ \mathcal{U}\ g))$
by *auto*
have *5*: $\vdash\ \neg(\#True\ \mathcal{U}\ g) = \Box\ (\neg\ g)$
using *BoxEqvFalseRelease NotUntilRelease inteq-reflection* **by** *fastforce*
from *5 4 3 1* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilBoxImp*:
$\vdash f\ \mathcal{U}\ (\Box\ g) \longrightarrow \Box(f\ \mathcal{U}\ g)$
proof −

have *1*: $\vdash f\ \mathcal{U}\ \Box g \longrightarrow f\ \mathcal{U}\ g$
by (*meson BoxElim BoxGen MP UntilRightMono*)
have *2*: $\vdash$ *wnext* $(f\ \mathcal{U}\ \Box g) = (empty \vee \bigcirc (f\ \mathcal{U}\ \Box g))$
by (*meson WnextEqvEmptyOrNext*)
have *3*: $\vdash \Box\ g = (g \wedge$ *wnext* $(\Box g))$
by (*metis* (*no-types*) *BoxEqvAndWnextBox*)
have *4*: $\vdash f\ \mathcal{U}\ \Box g = (\Box g \vee f \wedge \bigcirc (f\ \mathcal{U}\ \Box g))$
by (*meson UntilNextUntil*)
have *5*: $\vdash g \wedge$ *wnext* $(\Box g) \longrightarrow empty \vee \bigcirc (f\ \mathcal{U}\ \Box g)$
by (*metis NextUntil Prop01 Prop05 Prop08 UntilIntro WnextEqvEmptyOrNext int-iffD1 inteq-reflection*)
have *6*: $\vdash$ *more* $\wedge f\ \mathcal{U}\ \Box g \longrightarrow$ *empty* $\vee \bigcirc (f\ \mathcal{U}\ \Box g)$
using *5 3 4* **by** *fastforce*
have *7*: $\vdash$ *more* $\wedge f\ \mathcal{U}\ \Box g \longrightarrow \bigcirc (f\ \mathcal{U}\ \Box g)$
using *2 6 WnextAndMoreEqvNext* **by** *fastforce*
from *1 7* **show** *?thesis* **using** *BoxIntro*[*of LIFT*$(f\ \mathcal{U}\ (\Box\ g))$ *LIFT*$(f\ \mathcal{U}\ g)$]
by *auto*
qed

lemma *UntilBoxEqvBox*:
$\vdash f\ \mathcal{U}\ (\Box\ f) = \Box\ f$
proof −
have *1*: $\vdash\ f\ \mathcal{U}\ (\Box\ f) \longrightarrow \Box(f\ \mathcal{U}\ f)$
using *UntilBoxImp*[*of f f*] **by** *auto*
have *2*: $\vdash \Box(f\ \mathcal{U}\ f) = \Box\ f$
by (*simp add*: *BoxEqvBox UntilIdempotent*)
have *3*: $\vdash \Box\ f \longrightarrow f\ \mathcal{U}\ (\Box\ f)$
by (*simp add*: *UntilIntro*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilRightStrengthen*:
$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{U}\ h)$
by (*meson BoxGen MP OrImpUntil UntilRightMono*)

lemma *UntilLeftStrengthen*:
$\vdash (f \wedge g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ g)\ \mathcal{U}\ \ h$
by (*simp add*: *OrImpUntil UntilImpUntil*)

lemma *UntilLeftAndOrder*:
$\vdash (f \wedge g)\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{U}\ h)$
by (*metis Prop12 UntilIdempotent UntilImpUntil UntilIntro inteq-reflection*)

lemma *UntilFrameNext*:
$\vdash \Box\ f \longrightarrow (\bigcirc\ g \longrightarrow \bigcirc\ (f\ \mathcal{U}\ g))$
by (*simp add*: *NextImpNext Prop01 Prop05 Prop09 UntilIntro*)

lemma *UntilFrameDiamond*:
$\vdash \Box\ f \longrightarrow (\Diamond\ g \longrightarrow \Diamond\ (f\ \mathcal{U}\ g))$
by (*meson NowImpDiamond Prop09 UntilAndImp lift-imp-trans*)

lemma *UntilFrameBox*:
$\vdash \Box\ f \longrightarrow (\Box\ g \longrightarrow \Box\ (f\ \mathcal{U}\ g))$
by (*simp add*: *BoxAndBoxImpBoxRule OrImpUntil Prop09*)

lemma *UntilImpNot*:
$\vdash f\ \mathcal{U}\ g \longrightarrow (f \wedge \neg g)\ \mathcal{U}\ g$
proof −
have *1*: $\vdash f\ \mathcal{U}\ g \longrightarrow \Diamond\ g$
by (*simp add*: *UntilImpDiamond*)
have *2*: $\vdash \Diamond\ g = \#True\ \mathcal{U}\ g$
by (*simp add*: *DiamondEqvTrueUntil*)
have *3*: $\vdash \#True\ \mathcal{U}\ g \longrightarrow (\neg g)\ \mathcal{U}\ g$
by (*simp add*: *TrueUntilImpNotUntil*)
have *4*: $\vdash (f\ \mathcal{U}\ g \wedge (\neg g)\ \mathcal{U}\ g) = (f \wedge \neg g)\ \mathcal{U}\ g$
using *UntilAndDist* **by** *fastforce*
show *?thesis*
by (*meson 1 2 3 4 Prop10 int-iffD1 lift-imp-trans*)
qed

lemma *UntilAndRule*:
$\vdash f\ \mathcal{U}\ g = (f \wedge \neg g)\ \mathcal{U}\ g$
proof −
have *1*: $\vdash (f \wedge \neg g)\ \mathcal{U}\ g \longrightarrow f\ \mathcal{U}\ g$
using *UntilAndDist* **by** *fastforce*
show *?thesis* **by** (*simp add*: *1 UntilImpNot int-iffI*)
qed

lemma *UntilWait*:
$\vdash f\ \mathcal{U}\ g = (f\ \mathcal{W}\ g \wedge \Diamond\ g)$
proof −
have *1*: $\vdash f\ \mathcal{U}\ g \longrightarrow f\ \mathcal{W}\ g \wedge \Diamond\ g$
by (*simp add*: *Prop05 Prop12 UntilImpDiamond wait-d-def*)
have *2*: $\vdash (f\ \mathcal{W}\ g \wedge \Diamond\ g)\ = ((\Box\ f \vee f\ \mathcal{U}\ g) \wedge \Diamond\ g)$
by (*auto simp add*: *wait-d-def*)
have *3*: $\vdash ((\Box\ f \vee f\ \mathcal{U}\ g) \wedge \Diamond\ g) = (\ (\Box\ f \wedge \Diamond g) \vee (f\ \mathcal{U}\ g \wedge \Diamond\ g))$
by *auto*
have *4*: $\vdash (\Box\ f \wedge \Diamond g) \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *UntilAndImp*)
have *5*: $\vdash (f\ \mathcal{U}\ g \wedge \Diamond\ g) \longrightarrow f\ \mathcal{U}\ g$
by *auto*
show *?thesis*
using *1 2 4* **by** *fastforce*
qed

lemma *WaitLeftDistAnd*:
$\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow f\ \mathcal{W}\ g \wedge f\ \mathcal{W}\ h$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow f\ \mathcal{W}\ g$
unfolding *wait-d-def*

by (*metis Prop08 Prop12 UntilAbsorp-a UntilLeftDistAnd int-iffD1 inteq-reflection*)
have *2*: $\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow\ f\ \mathcal{W}\ h$
unfolding *wait-d-def*
by (*metis Prop08 Prop12 UntilAbsorp-a UntilLeftDistAnd int-iffD1 inteq-reflection*)
show *?thesis* **by** (*simp add*: *1 2 Prop12*)
qed

lemma *WaitRightDistAnd*:
$\vdash (f \wedge g)\ \mathcal{W}\ h = (f\ \mathcal{W}\ h \wedge g\ \mathcal{W}\ h)$
proof −
have *1*: $\vdash \Box(f \wedge g) = (\Box\ f \wedge \Box\ g)$
by (*metis BoxAndBoxEqvBoxRule inteq-reflection lift-and-com*)
have *2*: $\vdash (f \wedge g)\ \mathcal{U}\ h = (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)$
by (*simp add*: *UntilAndDist*)
have *3*: $\vdash ((\Box\ f \wedge \Box\ g)\ ) \longrightarrow ((\Box f \vee f\ \mathcal{U}\ h) \wedge (\Box g \vee g\ \mathcal{U}\ h))$
by (*simp add*: *intI*)
have *4*: $\vdash (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h) \longrightarrow ((\Box f \vee f\ \mathcal{U}\ h) \wedge (\Box g \vee g\ \mathcal{U}\ h))$
by *auto*
have *5*: $\vdash ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)) \longrightarrow ((\Box f \vee f\ \mathcal{U}\ h) \wedge (\Box g \vee g\ \mathcal{U}\ h))$
using *3 4* **by** *fastforce*
have *6*: $\vdash \Box\ f \wedge \Box\ g \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
by *auto*
have *7*: $\vdash \Box f \wedge g\ \mathcal{U}\ h \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
by (*metis 2 Prop05 Prop12 UntilAlwaysAndDist UntilLeftDistAnd inteq-reflection lift-imp-trans*)
have *8*: $\vdash f\ \mathcal{U}\ h \wedge \Box\ g \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
by (*metis Prop05 Prop08 Prop12 UntilAlwaysAndDist UntilAndDist UntilLeftDistAnd inteq-reflection lift-and-com*)
have *9*: $\vdash f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
by *auto*
have *10*: $\vdash ((\Box f \vee f\ \mathcal{U}\ h) \wedge (\Box g \vee g\ \mathcal{U}\ h)) \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
using *6 7 8 9* **by** *fastforce*
have *11*: $\vdash ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)) = ((\Box f \vee f\ \mathcal{U}\ h) \wedge (\Box g \vee g\ \mathcal{U}\ h))$
using *5 10* **by** *auto*
have *12*: $\vdash\ (\Box(f \wedge g) \vee (f \wedge g)\ \mathcal{U}\ h) = ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
using *1 2* **by** *fastforce*
show *?thesis* **unfolding** *wait-d-def* **using** *11 12*
by (*meson Prop04 UntilIdempotent*)
qed

lemma *WaitAndRule*:
$\vdash f\ \mathcal{W}\ g = (f \wedge \neg\ g)\ \mathcal{W}\ g$
proof −
have *1*: $\vdash (f\ \mathcal{W}\ g \wedge (\neg g\ )\ \mathcal{W}\ g) = (f \wedge \neg\ g)\ \mathcal{W}\ g$
by (*meson Prop11 WaitRightDistAnd*)
have *2*: $\vdash (\neg g\ )\ \mathcal{W}\ g$
by (*metis* (*no-types*, *opaque-lifting*) *DiamondEqvTrueUntil FalseUntil UntilAndRule UntilExclMid always-d-def int-simps*(*17*) *int-simps*(*4*) *inteq-reflection wait-d-def*)
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *WaitUntilb*:
$\vdash f\ \mathcal{W}\ \ g = (\Box\ (\ f \wedge \neg\ g) \vee f\ \mathcal{U}\ g)$
proof −
have *1*: $\vdash f\ \mathcal{W}\ g = (f \wedge \neg\ g)\ \mathcal{W}\ g$
by (*simp add*: *WaitAndRule*)
have *2*: $\vdash (f \wedge \neg\ g)\ \mathcal{W}\ g = (\Box\ (f \wedge \neg\ g) \vee (f \wedge \neg\ g)\ \mathcal{U}\ g)$
by (*auto simp add*: *wait-d-def*)
have *3*: $\vdash (f \wedge \neg\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*meson Prop11 UntilAndRule*)
show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *UntilNotDistWait*:
$\vdash (\neg\ (f\ \mathcal{U}\ g)) = (\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
proof −
have *1*: $\vdash (\neg\ ((\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g))) = (\neg(\neg\ f \wedge \neg\ g))\ \mathcal{U}\ (\ \neg\ (\neg\ g) \wedge \neg\ (\neg\ f \wedge \neg\ g))$
using *WaitNotDistUntil* **by** *blast*
have *2*: $\vdash (\neg(\neg\ f \wedge \neg\ g)) = (f \vee g)$
by *auto*
have *3*: $\vdash (\ \neg\ (\neg\ g) \wedge \neg\ (\neg\ f \wedge \neg\ g)) = g$
by *auto*
have *4*: $\vdash (\neg(\neg\ f \wedge \neg\ g))\ \mathcal{U}\ (\ \neg\ (\neg\ g) \wedge \neg\ (\neg\ f \wedge \neg\ g)) =$
$(f \vee g)\ \mathcal{U}\ g$
using *2 3 UntilEqvUntil* **by** *blast*
have *5*: $\vdash (f \vee g)\ \mathcal{U}\ g = ((f \vee g) \wedge \neg g)\ \mathcal{U}\ g$
by (*simp add*: *UntilAndRule*)
have *6*: $\vdash ((f \vee g) \wedge \neg g) = (f \wedge \neg\ g)$
by *auto*
have *7*: $\vdash ((f \vee g) \wedge \neg g)\ \mathcal{U}\ g = (f \wedge \neg g)\ \mathcal{U}\ g$
using *6 inteq-reflection* **by** *fastforce*
have *8*: $\vdash (f \wedge \neg g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*meson Prop11 UntilAndRule*)
have *9*: $\vdash f\ \mathcal{U}\ g = (\neg\ ((\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)))$
using *1 4 5 7 8* **by** *fastforce*
show *?thesis* **using** *9* **by** *auto*
qed

lemma *UntilImpWait*:
$\vdash f\ \mathcal{U}\ g \longrightarrow f\ \mathcal{W}\ g$
by (*meson Prop03 WaitUntilb*)

lemma *WaitAndDist*:
$\vdash (\Box\ f \wedge g\ \mathcal{W}\ h) \longrightarrow (f \wedge g)\ \mathcal{W}\ (f \wedge h)$
proof −
have *1*: $\vdash (\Box\ f \wedge g\ \mathcal{W}\ h) = (\Box\ f \wedge (\Box\ g \vee g\ \mathcal{U}\ h))$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash (\Box\ f \wedge (\Box\ g \vee g\ \mathcal{U}\ h)) = ((\Box f \wedge \Box g) \vee (\Box\ f \wedge g\ \mathcal{U}\ h))$
by *auto*

have *3*: $\vdash (\Box f \wedge \Box g) = \Box(f \wedge g)$
by (*simp add*: *BoxAndBoxEqvBoxRule*)
have *4*: $\vdash (\Box\ f \wedge g\ \mathcal{U}\ h) \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h)$
by (*simp add*: *UntilAlwaysAndDist*)
have *5*: $\vdash ((\Box f \wedge \Box g) \vee (\Box\ f \wedge g\ \mathcal{U}\ h)) \longrightarrow \Box(f \wedge g) \vee (f \wedge g)\ \mathcal{U}\ (f \wedge h)$
using *3 4* **by** *fastforce*
have *6*: $\vdash (\Box(f \wedge g) \vee (f \wedge g)\ \mathcal{U}\ (f \wedge h)) = (f \wedge g)\ \mathcal{W}\ (f \wedge h)$
by (*auto simp add*: *wait-d-def*)
show *?thesis*
using *1 5 6* **by** *fastforce*
qed

lemma *WaitDiamondOr*:
$\vdash f\ \mathcal{W}\ (\Diamond\ g) = (\Box\ f \vee \Diamond\ g)$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (\Diamond\ g) = (\Box\ f \vee f\ \mathcal{U}\ (\Diamond\ g))$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash f\ \mathcal{U}\ (\Diamond\ g) = \Diamond\ g$
by (*simp add*: *UntilAbsorpDiamond*)
show *?thesis* **using** *1 2 Prop06* **by** *blast*
qed

lemma *WaitBoxImp*:
$\vdash f\ \mathcal{W}\ (\Box\ g) \longrightarrow \Box\ (\ f\ \mathcal{W}\ g)$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (\Box\ g) = (\Box\ f \vee f\ \mathcal{U}\ (\Box\ g))$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash \Box\ f = \Box\ (\Box\ f)$
by (*simp add*: *BoxEqvBoxBox*)
have *3*: $\vdash f\ \mathcal{U}\ (\Box\ g) \longrightarrow \Box(f\ \mathcal{U}\ g)$
by (*simp add*: *UntilBoxImp*)
have *4*: $\vdash (\Box\ f \vee f\ \mathcal{U}\ (\Box\ g)) \longrightarrow (\Box\ (\Box\ f) \vee \Box(f\ \mathcal{U}\ g))$
using *2 3* **by** *fastforce*
have *5*: $\vdash \Box\ (\Box\ f) \longrightarrow \Box(\ \Box\ f \vee f\ \mathcal{U}\ g)$
by (*metis BoxImpBoxRule Prop08 UntilIdempotent UntilIntro int-simps*(*11*) *int-simps*(*25*)
inteq-reflection)
have *6*: $\vdash \Box(f\ \mathcal{U}\ g) \longrightarrow \Box(\ \Box\ f \vee f\ \mathcal{U}\ g)$
by (*metis BoxImpBoxRule UntilImpWait wait-d-def*)
have *7*: $\vdash (\Box\ (\Box\ f) \vee \Box(f\ \mathcal{U}\ g)) \longrightarrow \Box(\ \Box\ f \vee f\ \mathcal{U}\ g)$
using *5 6* **by** *fastforce*
have *6*: $\vdash \Box(\ \Box\ f \vee f\ \mathcal{U}\ g) = \Box\ (\ f\ \mathcal{W}\ g)$
by (*simp add*: *wait-d-def*)
show *?thesis*
by (*metis 4 7 lift-imp-trans wait-d-def*)
qed

lemma *WaitAbsorbBox*:
$\vdash f\ \mathcal{W}\ (\Box\ f) = \Box\ f$
by (*metis Prop02 Prop11 UntilBoxEqvBox UntilImpWait inteq-reflection wait-d-def*)

lemma *BoxImpWait*:
$\vdash \Box\ f \longrightarrow f\ \mathcal{W}\ g$
by (*auto simp add*: *wait-d-def*)

lemma *WaitDistNext*:
$\vdash \bigcirc\ (\ f\ \mathcal{W}\ g) = (\bigcirc\ f)\ \mathcal{W}\ (\bigcirc\ g)$
— nitpick finds counterexample, does not hold because of finite intervals
oops

lemma *WaitDistNextInfinite*:
$\vdash$ *inf* $\longrightarrow \bigcirc\ (\ f\ \mathcal{W}\ g) = (\bigcirc\ f)\ \mathcal{W}\ (\bigcirc\ g)$
proof −
have *1*: $\vdash \bigcirc\ (\Box\ f \vee f\ \mathcal{U}\ g) \longrightarrow (\bigcirc\ (\Box\ f) \vee \bigcirc(f\ \mathcal{U}\ g))$
by (*simp add*: *ChopOrImpRule next-d-def*)
have *2*: $\vdash (\bigcirc\ (\Box\ f) \vee \bigcirc(f\ \mathcal{U}\ g)) \longrightarrow \bigcirc\ (\Box\ f \vee f\ \mathcal{U}\ g)$
by (*metis BoxImpWait NextImpNext Prop02 UntilImpWait wait-d-def*)
have *21*: $\vdash \bigcirc\ (\Diamond(\neg\ f)) = \Diamond(\bigcirc(\neg f))$
by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopSkipEqvSkipChopFinite inteq-reflection next-d-def sometimes-d-def*)
have *22*: $\vdash (\neg(\ \bigcirc\ f)) = (empty \vee \bigcirc(\ \neg f))$
using *WnextEqvEmptyOrNext*[*of LIFT*($\neg f$)] **unfolding** *wnext-d-def* **by** *simp*
have *23*: $\vdash$ *inf* $\longrightarrow \bigcirc(\neg f) = (\neg(\ \bigcirc\ f))$
using *22 MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*
have *24*: $\vdash$ *inf* $\longrightarrow\ \Diamond(\bigcirc(\neg f)) = \Diamond(\neg(\ \bigcirc\ f))$
unfolding *sometimes-d-def* **using** *23*
InfRightChopEqvChop[*of LIFT*($\bigcirc(\neg f)$) *LIFT*($(\neg\ (\bigcirc\ f))$) *LIFT*(*finite*)]
by *simp*
have *25*: $\vdash$ *inf* $\longrightarrow \bigcirc\ (\Diamond(\neg\ f)) = \Diamond(\neg(\ \bigcirc\ f))$
using *21 24* **by** *fastforce*
have *26*: $\vdash$ *inf* $\longrightarrow\ (empty \vee \bigcirc(\Diamond(\neg\ f))) = \bigcirc\ (\Diamond(\neg\ f))$
using *MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*
have *27*: $\vdash$ *inf* $\longrightarrow$ *wnext* $(\Diamond(\neg\ f)) = \bigcirc\ (\Diamond(\neg\ f))$
by (*metis 26 WnextEqvEmptyOrNext inteq-reflection*)
have *28*: $\vdash$ *inf* $\longrightarrow (\neg\ (\bigcirc\ (\neg\ \Diamond(\neg\ f)))) = \Diamond(\neg\ \bigcirc\ f)$
using *21 24 27* **unfolding** *wnext-d-def* **by** *fastforce*
have *3*: $\vdash$ *inf* $\longrightarrow \bigcirc\ (\Box\ f) = \Box\ (\bigcirc\ f)$
unfolding *always-d-def* **using** *28* **by** *fastforce*
have *4*: $\vdash \bigcirc(f\ \mathcal{U}\ g) = \bigcirc\ f\ \mathcal{U}\ \bigcirc\ g$
by (*simp add*: *NextUntil*)
have *5*: $\vdash \bigcirc\ (\Box\ f \vee f\ \mathcal{U}\ g) = (\bigcirc\ (\Box\ f) \vee \bigcirc(f\ \mathcal{U}\ g))$
using *1 2 int-iffI* **by** *blast*
have *6*: $\vdash$ *inf* $\longrightarrow \bigcirc\ (\ f\ \mathcal{W}\ g) = \bigcirc\ (\Box\ f \vee f\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *7*: $\vdash$ *inf* $\longrightarrow \bigcirc\ (\Box\ f \vee f\ \mathcal{U}\ g) = (\bigcirc\ (\Box\ f) \vee \bigcirc(f\ \mathcal{U}\ g))$
using *5* **by** *auto*
have *8*: $\vdash$ *inf* $\longrightarrow (\bigcirc\ (\Box\ f) \vee \bigcirc(f\ \mathcal{U}\ g)) = (\Box\ (\bigcirc\ f) \vee \bigcirc\ f\ \mathcal{U}\ \bigcirc\ g)$
using *4 3* **by** *fastforce*
show *?thesis*
by (*metis 5 8 inteq-reflection wait-d-def*)

qed

lemma *WnextAlwaysEqvAlwaysWnext*:
$\vdash$ *finite* $\longrightarrow$ *wnext* $(\Box\ f) = \Box(\mathit{wnext}\ f)$
by (*metis* (*mono-tags*, *lifting*) *BoxEqvFiniteYields FiniteChopSkipEqvSkipChopFinite SkipYieldsEqvWeakNext YieldsYieldsEqvChopYields intI inteq-reflection unl-lift2*)

lemma *WaitExpand*:
$\vdash f\ \mathcal{W}\ g = (g \vee (f \wedge \bigcirc(f\ \mathcal{W}\ g)))$
— nitpick finds counterexample, does not hold because of finite intervals
oops

lemma *WaitExpand*:
$\vdash f\ \mathcal{W}\ g = (g \vee (f \wedge \mathit{wnext}(f\ \mathcal{W}\ g)))$
proof −
have *1*: $\vdash\ f\ \mathcal{W}\ g = (\Box\ f \vee f\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash \Box\ f = (f \wedge \mathit{wnext}(\Box\ f))$
by (*simp add*: *BoxEqvAndWnextBox*)
have *3*: $\vdash f\ \mathcal{U}\ g = (g \vee (f \wedge \bigcirc\ (f\ \mathcal{U}\ g)))$
using *UntilNextUntil* **by** *blast*
have *4*: $\vdash (f \wedge \mathit{wnext}(\Box\ f)) = (f \wedge (\mathit{empty} \vee \bigcirc\ (\Box\ f)))$
using *2 BoxEqvAndEmptyOrNextBox*[*of f*] **by** *fastforce*
have *5*: $\vdash\ \mathit{wnext}(f\ \mathcal{W}\ g) = (\mathit{empty} \vee \bigcirc\ (f\ \mathcal{W}\ g))$
using *WnextEqvEmptyOrNext* **by** *blast*
have *6*: $\vdash (f \wedge (\mathit{empty} \vee \bigcirc\ (\Box\ f))) = (\ (f \wedge \mathit{empty}) \vee (f \wedge \bigcirc\ (\Box\ f)))$
by *auto*
have *7*: $\vdash ((\ (f \wedge \mathit{empty}) \vee (f \wedge \bigcirc\ (\Box\ f))) \vee$
$(g \vee (f \wedge \bigcirc\ (f\ \mathcal{U}\ g)))\ ) =$
$(g \vee (f \wedge (\mathit{empty} \vee \bigcirc\ (\Box f)\ \ \vee \bigcirc\ (f\ \mathcal{U}\ g)\ )\ ))$
by *auto*
have *8*: $\vdash (\bigcirc\ (\Box f)\ \ \vee \bigcirc\ (f\ \mathcal{U}\ g)) = \bigcirc(\Box f \vee f\ \mathcal{U}\ g)$
by (*metis ChopOrEqv Prop11 next-d-def*)
show *?thesis*
by (*metis 1 2 3 4 5 6 7 8 inteq-reflection*)
qed

lemma *InfImpWnextEqnNext*:
$\vdash \mathit{inf} \longrightarrow \mathit{wnext}\ f = \bigcirc\ f$
proof −
have *1*: $\vdash \mathit{wnext}\ f = (\mathit{empty} \vee \bigcirc\ f)$
by (*simp add*: *WnextEqvEmptyOrNext*)
have *2*: $\vdash \mathit{inf} \longrightarrow \mathit{more}$
using *MoreAndInfEqvInf* **by** *auto*
have *3*: $\vdash \mathit{inf} \longrightarrow (\mathit{empty} \vee \bigcirc\ f) =\ \bigcirc\ f$
unfolding *empty-d-def* **using** *2* **by** *fastforce*
show *?thesis*
by (*metis 1 3 inteq-reflection*)
qed

lemma *WaitExpandInfinite*:
$\vdash$ *inf* $\longrightarrow$ $f\ \mathcal{W}\ g = (g \vee (f \wedge \bigcirc(f\ \mathcal{W}\ g)))$
proof −
have *1*: $\vdash f\ \mathcal{W}\ g = (g \vee (f \wedge$ *wnext*$(f\ \mathcal{W}\ g)))$
using *WaitExpand* **by** *blast*
have *2*: $\vdash$ *inf* $\longrightarrow$ *wnext*$(f\ \mathcal{W}\ g) = \bigcirc\ (f\ \mathcal{W}\ g)$
using *InfImpWnextEqnNext* **by** *blast*
have *3*: $\vdash$ *inf* $\longrightarrow (g \vee (f \wedge$ *wnext*$(f\ \mathcal{W}\ g))) = (g \vee (f \wedge \bigcirc(f\ \mathcal{W}\ g)))$
using *2* **by** *auto*
show *?thesis* **using** *1 3* **by** *fastforce*
qed

lemma *WaitExclMid*:
$\vdash f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ (\neg\ g)$
using *WaitExpand*
proof −
have *1*: $\vdash f\ \mathcal{W}\ g = (g \vee f \wedge$ *wnext* $(f\ \mathcal{W}\ g))$
by (*simp add*: *WaitExpand*)
have *2*: $\vdash f\ \mathcal{W}\ (\neg\ g) = ((\neg\ g) \vee f \wedge$ *wnext* $(\ f\ \mathcal{W}\ (\neg\ g)))$
by (*simp add*: *WaitExpand*)
have *3*: $\vdash (f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ (\neg\ g)) =$
$(\ (g \vee f \wedge$ *wnext* $(f\ \mathcal{W}\ g)) \vee ((\neg\ g) \vee f \wedge$ *wnext* $(\ f\ \mathcal{W}\ (\neg\ g))))$
using *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** *fastforce*
qed

lemma *WaitleftZero*:
$\vdash \#True\ \mathcal{W}\ g$
by (*meson BoxGen BoxImpWait MP TrueW*)

lemma *WaitLeftDistOr*:
$\vdash f\ \mathcal{W}\ (g \vee h) = (f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ h)$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (g \vee h) = (\ \Box\ f \vee f\ \mathcal{U}\ (g \vee h))$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash (f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ h) = ((\Box\ f \vee f\ \mathcal{U}\ g) \vee (\Box\ f \vee f\ \mathcal{U}\ h))$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash f\ \mathcal{U}\ (g \vee h) = (f\ \mathcal{U}\ g \vee f\ \mathcal{U}\ h)$
by (*simp add*: *UntilOrDist*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *WaitRightDistOr*:
$\vdash f\ \mathcal{W}\ h \vee g\ \mathcal{W}\ h \longrightarrow (f \vee g)\ \mathcal{W}\ h$
proof −
have *0*:$\vdash \Box\ g \longrightarrow \Box\ (f \vee g)$
by (*simp add*: *BoxImpBoxRule intI*)
have *1*: $\vdash \Box f \longrightarrow \Box\ (f \vee g)$
by (*simp add*: *BoxImpBoxRule intI*)
have *11*: $\vdash \Box\ f \vee \Box\ g \longrightarrow \Box\ (f \vee g)$

using *0 1 Prop02* **by** *blast*
have *2*: $\vdash\ f\ \mathcal{W}\ h = (\Box\ f \vee f\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *3*:$\vdash (f \vee g)\ \mathcal{W}\ h = (\Box\ (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *4*: $\vdash g\ \mathcal{W}\ h = (\Box g\ \vee g\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *5*: $\vdash f\ \mathcal{U}\ h \vee g\ \mathcal{U}\ h \longrightarrow (f \vee g)\ \mathcal{U}\ h$
using *UntilRightDistOr* **by** *simp*
have *6*:$\vdash (f\ \mathcal{W}\ h \vee g\ \mathcal{W}\ h) = ((\Box\ f \vee \Box\ g) \vee (f\ \mathcal{U}\ h \vee g\ \mathcal{U}\ h))$
using *2 4* **by** *fastforce*
from *11 5 6 3* **show** *?thesis*
by (*meson BoxImpWait Prop02 Prop11 UntilImpWait lift-imp-trans*)
qed

lemma *WaitOrRule*:
$\vdash f\ \mathcal{W}\ g = (f \vee g)\ \mathcal{W}\ g$
proof −
have *1*: $\vdash f\ \mathcal{W}\ g \longrightarrow (f \vee g)\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *Prop03 Prop10 UntilAbsorp-a WaitNotDistUntil int-iffD1 int-simps*(*14*) *int-simps*(*32*) *int-simps*(*33*) *inteq-reflection*)
have *2*: $\vdash (f \vee g)\ \mathcal{W}\ g \longrightarrow f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *Prop03 Prop10 WaitNotDistUntil int-iffD2 int-simps*(*14*) *int-simps*(*32*) *int-simps*(*33*) *inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilOrRule*:
$\vdash f\ \mathcal{U}\ g = (f \vee g)\ \mathcal{U}\ g$
by (*metis UntilWait WaitOrRule inteq-reflection*)

lemma *WaitRule*:
$\vdash (\neg f)\ \mathcal{W}\ f$
by (*metis BoxGen BoxImpWait MP WaitOrRule int-eq-true int-simps*(*29*) *inteq-reflection*)

lemma *UntilRule*:
$\vdash (\neg f)\ \mathcal{U}\ f = \Diamond f$
using *DiamondEqvTrueUntil UntilOrRule inteq-reflection* **by** *fastforce*

lemma *WaitImpRule*:
$\vdash (f \longrightarrow g)\ \mathcal{W}\ f$
proof −
have *1*: $\vdash (f \longrightarrow g)\ \mathcal{W}\ f = ((f \longrightarrow g) \vee f)\ \mathcal{W}\ f$
by (*simp add*: *WaitOrRule*)
have *2*: $\vdash (f \longrightarrow g) \vee f$
by *auto*
have *3*: $\vdash ((f \longrightarrow g) \vee f)\ \mathcal{W}\ f = \#True\ \mathcal{W}\ f$
by (*metis 1 2 int-eq-true inteq-reflection*)
show *?thesis*
using *1 3 WaitleftZero* **by** *fastforce*

qed

lemma *DiamondUntilImpRule*:
$\vdash \Diamond\ f \longrightarrow (f \longrightarrow g)\ \mathcal{U}\ f$
using *UntilWait WaitImpRule* **by** *fastforce*

lemma *WaitNotDist*:
$\vdash (\neg\ (f\ \mathcal{W}\ g)) = (f \wedge \neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg\ g)$
proof −
have *1*: $\vdash (\neg\ (f\ \mathcal{W}\ g)) = (\neg\ g\ )\ \mathcal{U}\ (\neg\ f \wedge \neg\ g)$
using *WaitNotDistUntil* **by** *blast*
have *2*: $\vdash (\neg\ g\ )\ \mathcal{U}\ (\neg\ f \wedge \neg\ g) = (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g))\ \mathcal{U}\ (\neg\ f \wedge \neg\ g)$
using *UntilAndRule* **by** *blast*
have *3*: $\vdash (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g)) = (f \wedge \neg\ g)$
by *auto*
have *4*: $\vdash (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g))\ \mathcal{U}\ (\neg\ f \wedge \neg\ g) = (f \wedge \neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg\ g)$
using *3 inteq-reflection* **by** *force*
show *?thesis* **using** *1 2 4* **by** *fastforce*
qed

lemma *UntilNotDist*:
$\vdash (\neg\ (f\ \mathcal{U}\ g)) = (f \wedge \neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
proof −
have *1*: $\vdash (\neg\ (f\ \mathcal{U}\ g)) = (\neg\ g\ )\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
using *UntilNotDistWait* **by** *blast*
have *2*: $\vdash\ (\neg\ g\ )\ \mathcal{W}\ (\neg\ f \wedge \neg\ g) = (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g))\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
by (*simp add*: *WaitAndRule*)
have *3*: $\vdash (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g)) = (f \wedge \neg\ g)$
by *auto*
have *4*: $\vdash (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g))\ \mathcal{W}\ (\neg\ f \wedge \neg\ g) = (f \wedge \neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
using *3 inteq-reflection* **by** *force*
show *?thesis* **using** *1 2 4* **by** *fastforce*
qed

lemma *UntilDuala*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = g\ \mathcal{W}\ (f \wedge g)$
proof −
have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{W}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
using *UntilNotDist* **by** *blast*
have *2*: $\vdash (\neg\ f \wedge g)\ \mathcal{W}\ (\ f \wedge g) = g\ \mathcal{W}\ (\ f \wedge g)$
using *1 UntilNotDistWait int-eq* **by** *fastforce*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *UntilDualb*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = (\neg\ f \wedge g)\ \mathcal{W}\ (f \wedge g)$
proof −
have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{W}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
using *UntilNotDist* **by** *blast*

show *?thesis*
  using *1* **by** *auto*
qed

lemma *WaitDuala*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = g\ \mathcal{U}\ (f \wedge g)$
proof −
 have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{U}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
  using *WaitNotDist* **by** *blast*
 have *2*: $\vdash (\neg\ f \wedge g)\ \mathcal{U}\ (\ f \wedge g) = g\ \mathcal{U}\ (\ f \wedge g)$
  using *1 WaitNotDistUntil int-eq* **by** *fastforce*
 show *?thesis*
  using *1 2* **by** *fastforce*
qed

lemma *WaitDualb*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = (\neg\ f \wedge g)\ \mathcal{U}\ (f \wedge g)$
proof −
 have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{U}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
  using *WaitNotDist* **by** *blast*
 show *?thesis* **using** *1* **by** *auto*
qed

lemma *WaitIdempotent*:
$\vdash f\ \mathcal{W}\ f = f$
by (*meson BoxElim Prop02 Prop12 UntilIdempotent UntilImpWait UntilIntro WaitUntilb int-iffD1 int-iffI lift-imp-trans*)

lemma *WaitRightZero*:
$\vdash f\ \mathcal{W}\ \#True$
by (*meson MP TrueW UntilImpWait UntilIntro*)

lemma *WaitLeftIdentity*:
$\vdash \#False\ \mathcal{W}\ g = g$
by (*metis* (*no-types*, *lifting*) *UntilAbsorp-c UntilNotDistWait WaitDuala WaitIdempotent WaitSRelease int-eq int-simps*(*17*) *int-simps*(*3*) *srelease-d-def*)

lemma *WaitImpOr*:
$\vdash f\ \mathcal{W}\ g \longrightarrow f \vee g$
 by (*metis Prop03 WaitIdempotent WaitLeftDistOr WaitOrRule inteq-reflection*)

lemma *BoxOrImpWait*:
$\vdash \Box(f \vee g) \longrightarrow f\ \mathcal{W}\ g$
using *BoxImpWait WaitOrRule* **by** *fastforce*

lemma *BoxImpImpWait*:
$\vdash \Box(\neg\ g \longrightarrow f) \longrightarrow f\ \mathcal{W}\ g$
proof −
 have *1*: $\vdash (\neg\ g \longrightarrow f) = (f \vee g)$
  by *auto*

have *2*: $\vdash \Box\ (\neg\ g \longrightarrow f) = \Box(f \vee g)$
using *1 BoxEqvBox* **by** *blast*
show *?thesis* **using** *2 BoxOrImpWait* **by** *fastforce*
qed

lemma *WaitInsertion*:
$\vdash g \longrightarrow f\ \mathcal{W}\ g$
by (*simp add*: *Prop05 UntilIntro wait-d-def*)

lemma *WaitFrameNext*:
$\vdash \Box\ f \longrightarrow (\bigcirc\ g \longrightarrow \bigcirc\ (f\ \mathcal{W}\ g))$
by (*simp add*: *NextImpNext Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameDiamond*:
$\vdash \Box\ f \longrightarrow (\Diamond\ g \longrightarrow \Diamond\ (f\ \mathcal{W}\ g))$
by (*simp add*: *DiamondImpDiamond Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameBox*:
$\vdash \Box\ f \longrightarrow (\Box\ g \longrightarrow \Box\ (f\ \mathcal{W}\ g))$
by (*meson BoxAndBoxImpBoxRule OrImpUntil Prop09 UntilImpWait lift-imp-trans*)

lemma *WaitInductiona*:
$\vdash \Box\ (f \longrightarrow (\bigcirc\ f \wedge g) \vee h) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ h)$
by (*simp add*: *UntilInduction-a wait-d-def*)

lemma *WaitInductionb*:
$\vdash \Box\ (f \longrightarrow \bigcirc\ f \vee g) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
by (*simp add*: *UntilInduction-b wait-d-def*)

lemma *WaitInductionc*:
$\vdash \Box(f \longrightarrow \bigcirc\ f) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
proof −
have *1*: $\vdash (f \longrightarrow \bigcirc\ f) \longrightarrow\ (f \longrightarrow wnext\ f)$
unfolding *wnext-d-def* **using** *NextImpNotNextNot*[*of f*] **by** *auto*
have *2*: $\vdash \Box(f \longrightarrow \bigcirc\ f) \longrightarrow \Box\ (f \longrightarrow wnext\ f)$
using *1 BoxImpBoxRule* **by** *blast*
show *?thesis* **by** (*meson 2 BoxImpWait BoxInduct Prop09 lift-imp-trans*)
qed

lemma *WaitInductiond*:
$\vdash \Box\ (f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
proof −
have *1*: $\vdash (f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow\ (f \longrightarrow\ wnext\ f)$
unfolding *wnext-d-def* **using** *NextImpNotNextNot*[*of f*] **by** *auto*
have *2*: $\vdash \Box(f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow \Box\ (f \longrightarrow\ wnext\ f)$
using *1 BoxImpBoxRule* **by** *blast*
show *?thesis* **by** (*meson 2 BoxImpWait BoxInduct Prop09 lift-imp-trans*)
qed

lemma *WaitAbsorba*:

$\vdash (f \vee f\ \mathcal{W}\ g) = (f \vee g)$
proof −
have *1*: $\vdash (f \vee f\ \mathcal{W}\ g) \longrightarrow (f \vee g)$
using *WaitImpOr* **by** *fastforce*
have *2*: $\vdash f \vee g \longrightarrow f \vee f\ \mathcal{W}\ g$
using *WaitInsertion* **by** *fastforce*
show *?thesis* **using** *1 2 int-iffI* **by** *blast*
qed

lemma *WaitAbsorbb*:
$\vdash (f\ \mathcal{W}\ g \vee g) = f\ \mathcal{W}\ g$
using *WaitInsertion*[*of g f*] **by** *auto*

lemma *WaitAbsorbc*:
$\vdash (f\ \mathcal{W}\ g \wedge g) = g$
using *WaitInsertion* **by** *fastforce*

lemma *WaitAbsorbd*:
$\vdash (f\ \mathcal{W}\ g \wedge (f \vee g)) = f\ \mathcal{W}\ g$
by (*meson Prop10 Prop11 WaitImpOr*)

lemma *WaitAbsorbe*:
$\vdash (f\ \mathcal{W}\ g \vee (f \wedge g)) = f\ \mathcal{W}\ g$
unfolding *wait-d-def*
using *UntilAbsorp-d* **by** *fastforce*

lemma *WaitLeftAbsorb*:
$\vdash f\ \mathcal{W}\ (f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *BoxEqvBoxBox UntilUntil WaitAbsorbBox WaitAbsorba WaitLeftDistOr inteq-reflection wait-d-def*)

lemma *WaitRightAbsorb*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{W}\ g = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *LeftUntilAbsorp Prop10 WaitInsertion WaitNotDistUntil int-iffD1 int-iffI int-simps*(*32*) *inteq-reflection*)

lemma *WaitBox*:
$\vdash \Box f = f\ \mathcal{W}\ \#False$
by (*metis* (*no-types*, *lifting*) *BoxGen DiamondNotEqvNotBox UntilAbsorpAndDiamond UntilAbsorp-c int-eq-true int-simps*(*2*) *int-simps*(*25*) *inteq-reflection wait-d-def*)

lemma *WaitDiamond*:
$\vdash \Diamond f = (\neg(\ (\neg f)\ \mathcal{W}\ \#False))$
using *DiamondNotEqvNotBox WaitBox* **by** *fastforce*

lemma *WaitImp*:
$\vdash f\ \mathcal{W}\ g \longrightarrow \Box f \vee \Diamond g$
by (*metis Prop08 UntilImpDiamond WaitAbsorbb WaitImpOr WaitRightAbsorb int-eq wait-d-def*)

lemma *WaitRightUntilAbsorb*:

$\vdash f\ \mathcal{W}\ (f\ \mathcal{U}\ g) = f\ \mathcal{W}\ g$
by (*metis UntilUntil WaitOrRule inteq-reflection wait-d-def*)

lemma *WaitLeftUntilAbsorb*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{W}\ g = f\ \mathcal{U}\ g$
by (*metis Prop11 RightUntilAbsorp UntilAbsorp-b UntilImpWait WaitImpOr inteq-reflection*)

lemma *UntilRightWaitAbsorb*:
$\vdash f\ \mathcal{U}\ (f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
using *UntilImpWait UntilIntro WaitLeftAbsorb* **by** *fastforce*

lemma *UntilLeftWaitAbsorb*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*metis UntilWait WaitRightAbsorb inteq-reflection*)

lemma *WaitDiamondAbsorb*:
$\vdash (\Diamond\ g)\ \mathcal{W}\ g = \Diamond\ g$
by (*metis DiamondEqvTrueUntil WaitLeftUntilAbsorb inteq-reflection*)

lemma *WaitAndBoxAbsorb*:
$\vdash (\Box\ f \wedge f\ \mathcal{W}\ g) = \Box\ f$
by (*meson BoxImpWait NotDiamondNotEqvBox Prop04 Prop10*)

lemma *WaitOrBoxAbsorb*:
$\vdash (\Box\ f \vee f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
by (*metis UntilRightWaitAbsorb WaitLeftAbsorb inteq-reflection wait-d-def*)

lemma *WaitAndBoxImpBox*:
$\vdash f\ \mathcal{W}\ g \wedge \Box\ (\neg\ g) \longrightarrow \Box\ f$
by (*metis* (*no-types*, *opaque-lifting*) *Prop02 Prop05 Prop07 Prop08 UntilIdempotent UntilImpDiamond UntilIntro always-d-def int-simps*(*25*) *int-simps*(*4*) *inteq-reflection wait-d-def*)

lemma *BoxImpUntilOrBox*:
$\vdash \Box\ f \longrightarrow f\ \mathcal{U}\ g \vee \Box\ (\neg\ g)$
proof −
have *1*: $\vdash (\Box\ f \longrightarrow f\ \mathcal{U}\ g \vee \Box\ (\neg\ g)) =$
$((\Box\ f \wedge \Diamond\ g) \longrightarrow f\ \mathcal{U}\ g)$
by (*auto simp add*: *always-d-def*)
have *2*: $\vdash (\Box\ f \wedge \Diamond\ g) \longrightarrow f\ \mathcal{U}\ g$
using *UntilAndImp* **by** *blast*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *NotBoxAndWaitImpDiamond*:
$\vdash \neg(\ \Box\ f) \wedge f\ \mathcal{W}\ g \longrightarrow \Diamond\ g$
using *WaitImp* **by** *fastforce*

lemma *DiamondImpNotBoxOrUntil*:
$\vdash \Diamond\ g \longrightarrow \neg\ (\Box\ f) \vee f\ \mathcal{U}\ g$

proof −
have *1*: $\vdash \Diamond\ g \wedge \Box\ f \longrightarrow f\ \mathcal{U}\ g$
using *UntilAndImp* **by** *fastforce*
show *?thesis* **using** *1* **by** *auto*
qed

lemma *WaitRightDistImp*:
$\vdash (f \longrightarrow g)\ \mathcal{W}\ h \longrightarrow (f\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ h)$
proof −
have *0*: $\vdash \Box(f \longrightarrow g) \wedge \Box f \longrightarrow \Box g$
by (*simp add*: *BoxAndBoxImpBoxRule intI*)
have *1*: $\vdash \Box(f \longrightarrow g) \wedge \Box f \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
using *0* **by** *auto*
have *2*: $\vdash \Box(f \longrightarrow g) \wedge f\ \mathcal{U}\ h \longrightarrow (\ (f \longrightarrow g) \wedge f)\ \mathcal{U}\ (\ (f \longrightarrow g) \wedge h)$
using *UntilAlwaysAndDist*[*of LIFT*$(f \longrightarrow g)$ *f h*] **by** *auto*
have *3*: $\vdash (f \longrightarrow g) \wedge f \longrightarrow g$
by *auto*
have *4*: $\vdash (f \longrightarrow g) \wedge h \longrightarrow h$
by *auto*
have *5*: $\vdash (\ (f \longrightarrow g) \wedge f)\ \mathcal{U}\ (\ (f \longrightarrow g) \wedge h) \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
by (*simp add*: *3 4 Prop05 UntilImpUntil*)
have *6*: $\vdash \Box(f \longrightarrow g) \wedge f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
using *2 5 lift-imp-trans* **by** *blast*
have *7*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Box f \vee f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
by (*simp add*: *1 6 Prop09 Prop12*)
have *8*: $\vdash \Box\ f \wedge (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f \wedge (f \longrightarrow g))\ \mathcal{U}\ (f \wedge h)$
by (*simp add*: *UntilAlwaysAndDist*)
have *9*: $\vdash f \wedge (f \longrightarrow g) \longrightarrow g$
by *auto*
have *10*: $\vdash f \wedge h \longrightarrow h$
by *auto*
have *11*: $\vdash (f \wedge (f \longrightarrow g))\ \mathcal{U}\ (f \wedge h) \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
by (*simp add*: *10 9 Prop05 UntilImpUntil*)
have *12*: $\vdash\ (f \longrightarrow g)\ \mathcal{U}\ h \wedge \Box f \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
using *8 11* **by** *fastforce*
have *13*:$\vdash (f \longrightarrow g)\ \mathcal{U}\ h \wedge f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
by (*metis 3 Prop05 UntilAndDist UntilImpUntil UntilIntro UntilUntil inteq-reflection*)
have *14*: $\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow \Box f \vee f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
by (*simp add*: *12 13 Prop09 Prop12*)
show *?thesis* **unfolding** *wait-d-def* **using** *7 14 Prop02* **by** *blast*
qed

lemma *WaitLeftMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (f\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ h)$
by (*meson BoxImpWait WaitRightDistImp lift-imp-trans*)

lemma *WaitRightMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{W}\ f \longrightarrow h\ \mathcal{W}\ g)$
proof −
have *1*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \longrightarrow h\ \mathcal{U}\ g)$

by (*simp add*: *UntilRightMono*)
have *2*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (\Box\ h \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
by *auto*
have *3*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
using *1* **by** *auto*
have *4*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (\Box\ h \vee h\ \mathcal{U}\ f \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *wait-d-def*)
qed

lemma *WaitStrengthen*:
$\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
proof −
have *1*: $\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g)$
by (*meson BoxAndBoxEqvBoxRule Prop01 Prop05 Prop11 WaitLeftMono lift-and-com lift-imp-trans*)
have *2*: $\vdash\ \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
by (*meson BoxElim BoxImpBoxBox BoxImpBoxRule Prop12 WaitRightMono lift-imp-trans*)
have *3*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g) \wedge (\ h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
using *1 2* **by** *fastforce*
have *4*: $\vdash (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g) \wedge (\ h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
by *auto*
from *3 4* **show** *?thesis* **by** *auto*
qed

lemma *WaitCatRule*:
$\vdash \Box(\ (f \longrightarrow g\ \mathcal{W}\ h) \wedge (h \longrightarrow g\ \mathcal{W}\ i)) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ i)$
proof −
have *1*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{W}\ \ h) \wedge (h \longrightarrow g\ \mathcal{W}\ \ i)) \longrightarrow \Box\ (f \longrightarrow g\ \mathcal{W}\ h)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *2*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{W}\ h) \wedge (h \longrightarrow g\ \mathcal{W}\ i)) \longrightarrow \Box\ (h \longrightarrow g\ \mathcal{W}\ i)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *3*: $\vdash \Box\ (h \longrightarrow g\ \mathcal{W}\ i) \longrightarrow \Box\ (\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ (g\ \mathcal{W}\ i))$
by (*metis BoxEqvBoxBox BoxImpBoxRule WaitRightMono inteq-reflection*)
have *4*: $\vdash \Box\ (\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ (g\ \mathcal{W}\ i)) \longrightarrow \Box(\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i)$
by (*metis BoxEqvBoxBox WaitLeftAbsorb int-iffD1 inteq-reflection*)
have *5*: $\vdash\ \Box\ (f \longrightarrow g\ \mathcal{W}\ h) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ h)$
by (*simp add*: *BoxElim*)
have *6*: $\vdash \Box(\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i) \longrightarrow (g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i)$
by (*simp add*: *BoxElim*)
have *7*: $\vdash (f \longrightarrow g\ \mathcal{W}\ h) \wedge (g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ i)$
by *auto*
have *8*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{W}\ h) \wedge (h \longrightarrow g\ \mathcal{W}\ i)) \longrightarrow$
$(f \longrightarrow g\ \mathcal{W}\ h) \wedge (g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i)$
using *1 2 3 4 5 6***by** *fastforce*
from *7 8* **show** *?thesis* **by** *auto*
qed

lemma *LeftUntilWaitImp*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{W}\ h \longrightarrow (f\ \mathcal{W}\ g)\ \mathcal{W}\ h$
by (*meson BoxGen MP UntilImpWait WaitLeftMono*)

lemma *RightWaitUntilImp*:
$\vdash f\ \mathcal{W}\ (g\ \mathcal{U}\ h) \longrightarrow f\ \mathcal{W}\ (g\ \mathcal{W}\ h)$
by (*meson BoxGen MP UntilImpWait WaitRightMono*)

lemma *RightUntilUntilImp*:
$\vdash f\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{W}\ h)$
by (*meson BoxGen MP UntilImpWait UntilRightMono*)

lemma *LeftUntilUntilImp*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{W}\ g)\ \mathcal{U}\ h$
by (*simp add*: *UntilImpUntil UntilImpWait*)

lemma *LeftUntilOrStrengthen*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{U}\ h \longrightarrow (f \vee g)\ \mathcal{U}\ h$
by (*simp add*: *UntilImpOr UntilImpUntil*)

lemma *LeftWaitOrStrengthen*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{W}\ h \longrightarrow (f \vee g)\ \mathcal{W}\ h$
by (*meson BoxGen MP WaitImpOr WaitLeftMono*)

lemma *RightWaitOrStrengthen*:
$\vdash f\ \mathcal{W}\ (g\ \mathcal{W}\ h) \longrightarrow f\ \mathcal{W}\ (g \vee h)$
by (*meson BoxGen MP WaitImpOr WaitRightMono*)

lemma *BoxImpBoxOr*:
$\vdash \Box f \longrightarrow \Box(f \vee g)$
by (*metis BoxEqvBoxBox BoxImpBoxRule BoxImpWait Prop12 WaitAbsorbd inteq-reflection*)

lemma *RightWaitOrOrder*:
$\vdash f\ \mathcal{W}\ (g\ \mathcal{W}\ h) \longrightarrow (f \vee g)\ \mathcal{W}\ h$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (g\ \mathcal{W}\ h) = (\Box f \vee f\ \mathcal{U}\ (\Box g \vee g\ \mathcal{U}\ h))$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash (f \vee g)\ \mathcal{W}\ h = (\Box (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash \Box f \longrightarrow (\Box (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
using *BoxImpBoxOr* **by** *fastforce*
have *4*: $\vdash f\ \mathcal{U}\ (\Box g \vee g\ \mathcal{U}\ h) = (f\ \mathcal{U}\ (\Box g) \vee f\ \mathcal{U}\ (g\ \mathcal{U}\ h))$
using *UntilOrDist* **by** *blast*
have *5*: $\vdash f\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow (\Box (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
by (*simp add*: *Prop05 UntilRightOr*)
have *6*: $\vdash f\ \mathcal{U}\ (\Box g) \longrightarrow (\Box (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
by (*metis BoxImpBoxRule BoxImpWait UntilBoxImp UntilImpOr lift-imp-trans wait-d-def*)
have *7*: $\vdash (f\ \mathcal{U}\ (\Box g) \vee f\ \mathcal{U}\ (g\ \mathcal{U}\ h)) \longrightarrow (\Box (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
using *5 6* **by** *fastforce*
show *?thesis*
using *1 2 3 4 7* **by** *fastforce*
qed

lemma *RightWaitAndOrder*:
$\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow f\ \mathcal{W}\ (g\ \mathcal{W}\ h)$
by (*metis Prop03 WaitAbsorbe WaitLeftDistOr inteq-reflection*)

lemma *UntilOrder*:
$\vdash ((\neg f)\ \mathcal{U}\ g \vee (\neg g)\ \mathcal{U}\ f) = \Diamond(f \vee g)$
proof −
have *1*: $\vdash ((\neg f)\ \mathcal{U}\ g \vee (\neg g)\ \mathcal{U}\ f) \longrightarrow \Diamond(f \vee g)$
using *UntilAbsorpAndDiamond UntilOrDist* **by** *fastforce*
have *2*: $\vdash \Diamond(f \vee g) = \#True\ \mathcal{U}\ (f \vee\ g)$
by (*metis DiamondEqvTrueUntil*)
have *3*: $\vdash \#True\ \mathcal{U}\ (f \vee\ g) = (\neg\ (f \vee g)\ )\ \mathcal{U}\ (f \vee g)$
using *2 UntilRule* **by** *fastforce*
have *4*: $\vdash (\neg\ (f \vee g)\ )\ \mathcal{U}\ (f \vee g) = (\neg f \wedge \neg g)\ \mathcal{U}\ (f \vee g)$
by (*metis UntilAbsorp-c int-eq int-simps*(*14*) *int-simps*(*33*))
have *5*: $\vdash (\neg f \wedge \neg g)\ \mathcal{U}\ (f \vee g) \longrightarrow (\neg\ f \wedge \neg g)\ \mathcal{U}\ f\ \vee (\neg\ f \wedge \neg g)\ \mathcal{U}\ g$
by (*simp add*: *UntilOrDist int-iffD1*)
have *6*: $\vdash (\neg\ f \wedge \neg g)\ \mathcal{U}\ f \longrightarrow (\neg\ g)\ \mathcal{U}\ f$
by (*metis UntilAndRule int-iffD2 inteq-reflection lift-and-com*)
have *7*: $\vdash (\neg\ f \wedge \neg g)\ \mathcal{U}\ g \longrightarrow (\neg\ f)\ \mathcal{U}\ g$
by (*metis UntilAndRule int-iffD2*)
have *8*: $\vdash (\neg\ f \wedge \neg g)\ \mathcal{U}\ f\ \vee (\neg\ f \wedge \neg g)\ \mathcal{U}\ g \longrightarrow (\neg\ f)\ \mathcal{U}\ g \vee (\neg\ g)\ \mathcal{U}\ f$
using *6 7* **by** *fastforce*
have *9*: $\vdash\ \Diamond(f \vee g) \longrightarrow (\neg\ f)\ \mathcal{U}\ g \vee (\neg\ g)\ \mathcal{U}\ f$
using *2 3 4 5 8* **by** *fastforce*
show *?thesis* **using** *1 9* **by** *fastforce*
qed

lemma *WaitOrder*:
$\vdash (\neg\ f)\ \mathcal{W}\ g \vee (\neg\ g)\ \mathcal{W}\ f$
proof −
have *1*: $\vdash (\neg\ f)\ \mathcal{W}\ g = (\Box\ (\neg\ f) \vee (\neg\ f)\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *2*:$\vdash (\neg\ g)\ \mathcal{W}\ f = (\Box\ (\neg\ g) \vee (\neg\ g)\ \mathcal{U}\ f)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash ((\Box\ (\neg\ f) \vee (\neg\ f)\ \mathcal{U}\ g) \vee (\Box\ (\neg\ g) \vee (\neg\ g)\ \mathcal{U}\ f)) =$
$(\ (\Box\ (\neg\ f) \vee \Box\ (\neg\ g)) \vee ((\neg\ f)\ \mathcal{U}\ g \vee (\neg\ g)\ \mathcal{U}\ f))$
by *auto*
have *4*: $\vdash ((\neg\ f)\ \mathcal{U}\ g \vee (\neg\ g)\ \mathcal{U}\ f) = \Diamond(f \vee g)$
using *UntilOrder* **by** *blast*
have *5*: $\vdash (\Box\ (\neg\ f) \vee \Box\ (\neg\ g)) = (\neg\ (\Diamond\ f) \vee \neg\ (\Diamond\ g))$
by (*simp add*: *always-d-def*)
have *6*: $\vdash \Diamond(f \vee g) = (\Diamond f \vee \Diamond\ g)$
by (*simp add*: *ChopOrEqv sometimes-d-def*)
have *7*: $\vdash (\Box\ (\neg\ f) \vee \Box\ (\neg\ g)) \vee \Diamond(f \vee g)$
using *5 6* **by** *fastforce*
show *?thesis*
using *1 2 4 7* **by** *fastforce*
qed

lemma *WaitImpOrder*:
$\vdash f\ \mathcal{W}\ g \wedge (\neg\ g)\ \mathcal{W}\ h \longrightarrow f\ \mathcal{W}\ h$
proof −
have *1*: $\vdash f\ \mathcal{W}\ g = (\Box f \vee f\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash f\ \mathcal{W}\ h = (\Box\ f \vee f\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash (\neg\ g)\ \mathcal{W}\ h = (\Box\ (\neg\ g) \vee (\neg\ g)\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *4*: $\vdash \Box\ f \wedge (\Box\ (\neg\ g) \vee (\neg\ g)\ \mathcal{U}\ h) \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
by *auto*
have *5*: $\vdash (\Box f \vee f\ \mathcal{U}\ g) \wedge \Box\ (\neg\ g) \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
using *1 WaitAndBoxImpBox* **by** *fastforce*
have *6*: $\vdash f\ \mathcal{U}\ g \wedge (\neg\ g)\ \mathcal{U}\ h \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
by (*simp add*: *Prop05 UntilNotImp*)
have *7*: $\vdash \Box\ f \wedge (\neg\ g)\ \mathcal{U}\ h \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
by *auto*
have *8*: $\vdash (\Box f \vee f\ \mathcal{U}\ g) \wedge\ \ (\neg\ g)\ \mathcal{U}\ h \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
using *6 7* **by** *fastforce*
have *9*: $\vdash (\Box f \vee f\ \mathcal{U}\ g) \wedge (\Box\ (\neg\ g) \vee (\neg\ g)\ \mathcal{U}\ h) \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
using *5 8* **by** *fastforce*
show *?thesis* **by** (*simp add*: *9 wait-d-def*)
qed

end

3.9 Pi operator

theory *Pi*
imports *Until Omega*
begin

This theory introduces the Pi operator [16, 13]. The Pi operator is defined in terms of the filter operator. We prove the soundness of the rules and axiom system. The until operator from Until.thy is used as there is a striking similarity of the expressiveness of the until and the Pi operator [13].

3.9.1 Definitions

definition *sfxfilt* :: *'a nellist* ⇒ (*'a*:: *world*) *formula* ⇒ *'a nellist nellist*
where *sfxfilt s f* = (*nfilter* (λ *σ*. *σ* ⊨*f*) (*ndropns s*))

definition *pifilt* :: *'a nellist* ⇒ (*'a*:: *world*) *formula* ⇒ *'a nellist*
where *pifilt s f* = (*nmap* (λ*s*. *nnth s 0*) (*sfxfilt s f*))

definition *pi-d* :: (*'a*:: *world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *pi-d F G* ≡ λ*s*. ((∃ *i* ≤ *nlength s*. (*ndropn i s*) ⊨ *F*) ∧ ((*pifilt s F*) ⊨ *G*))

syntax
-pi-d :: [*lift*,*lift*] ⇒ *lift* ((- Π -) [*84*,*84*] *83*)

syntax (*ASCII*)
-pi-d :: [*lift*,*lift*] ⇒ *lift* ((- *PI* -) [*84*,*84*] *83*)

translations
-pi-d ⇌ *CONST pi-d*

definition *upi-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *upi-d F G* ≡ *LIFT*(¬(*F* Π (¬ *G*)))

syntax
-upi-d :: [*lift*,*lift*] ⇒ *lift* ((- Π^u -) [*84*,*84*] *83*)

syntax (*ASCII*)
-upi-d :: [*lift*,*lift*] ⇒ *lift* ((- *UPI* -) [*84*,*84*] *83*)

translations
-upi-d ⇌ *CONST upi-d*

3.9.2 Semantic Lemmas

lemma *sfxfilter-help*:
(∃ *ys* ∈ *nset* (*ndropns xs*) . *f ys*) = (∃ *i* ≤ *nlength xs*. *f* (*ndropn i xs*))
using *in-nset-ndropns* **by** *auto*

lemma *pifiltinit-help*:
(∃ *y* ∈ *nset* (*xs*). *w* (*NNil y*)) = (∃ *i* ≤ *nlength xs*. *w* (*NNil* (*nnth xs i*)))
by (*metis in-nset-conv-nnth*)

lemma *sfxfilt-nnth*:
assumes (∃ *i* ≤ *nlength σ*. (*ndropn i σ*) ⊨ *f*)
i≤*nlength* (*sfxfilt σ f*)
shows (*nnth* (*sfxfilt σ f*) *i*) ⊨ *f*
using *assms* **by** (*metis in-nset-ndropns nkfilter-nnth-aa sfxfilt-def*)

lemma *pifilt-exists*:
assumes (∃ *i*≤*nlength σ*. *f* (*ndropn i σ*))
shows (∃ *i* ≤ *nlength*(*sfxfilt σ f*). (*nnth* (*sfxfilt σ f*) *i*) ⊨ *f*)
using *assms* **by** (*metis sfxfilt-nnth zero-enat-def zero-le*)

lemma *sfxfilt-pifilt-nlength*:
shows *nlength* (*pifilt σ f*) = *nlength* (*nmap* (λ*s*. *nnth s 0*) (*sfxfilt σ f*))
by (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt-nnth*:
assumes *j*≤ *nlength* (*pifilt σ f*)

shows *nnth (pifilt σ f) j = nnth (nmap (λs. nnth s 0) (sfxfilt σ f)) j*
using *assms* **by** (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt*:
shows *(nmap (λs. nnth s 0) (sfxfilt σ f)) = pifilt σ f*
by (*simp add*: *pifilt-def*)

lemma *sfxfilt-nlength-bound*:
assumes (∃ *i*≤*nlength xs. f (ndropn i xs)*)
shows *nlength (sfxfilt xs f) ≤ nlength xs*
using *assms*
by (*metis length-nfilter-le ndropns-nlength sfxfilt-def*)

lemma *pifilt-nlength-bound*:
assumes (∃ *i*≤*nlength σ. f (ndropn i σ)*)
shows *nlength (pifilt σ f) ≤ nlength σ*
using *assms* **by** (*simp add*: *pifilt-def sfxfilt-nlength-bound*)

lemma *sfxfilt-nlength-nnth-bound*:
assumes (∃ *i*≤*nlength σ. f (ndropn i σ)*)
j≤ *nlength (sfxfilt σ f)*
shows *nlength (nnth (sfxfilt σ f) j) ≤ nlength σ*
using *assms* **by** (*simp add*: *nfilter-ndropns-nnth-bound sfxfilt-def sfxfilter-help*)

lemma *sfxfilt-pifilt-nnth-ndropn*:
assumes (∃ *i ≤ nlength σ. (ndropn i σ) ⊨ f*)
j≤ *nlength (pifilt σ f)*
shows *nnth (sfxfilt σ f) j = ndropn (nnth (nkfilter f 0 (ndropns σ)) j) σ*
proof −
have *0*: ∃ *x*∈*nset (ndropns σ). f x*
by (*simp add*: *assms*(*1*) *sfxfilter-help*)
have *1*: *nnth (sfxfilt σ f) j = nnth (nfilter f (ndropns σ)) j*
by (*simp add*: *sfxfilt-def*)
have *2*: *nnth (nfilter f (ndropns σ)) j = nnth (ndropns σ) (nnth (nkfilter f 0 (ndropns σ)) j)*
using *nkfilter-nfilter*[*of ndropns σ λ s. s ⊨ f j 0*] *assms* **unfolding** *pifilt-def sfxfilt-def*
by (*simp add*: *0 nkfilter-nlength*)
have *30*: *enat (nnth (nkfilter f 0 (ndropns σ)) j) ≤ nlength σ*
using *0 nkfilter-upperbound*[*of (ndropns σ) f j 0*] *assms* **unfolding** *pifilt-def sfxfilt-def*
by (*metis gen-nlength-def ndropns-nlength nkfilter-nlength nlength-code nlength-nmap*)
have *3*: *nnth (ndropns σ) (nnth (nkfilter f 0 (ndropns σ)) j) =*
ndropn (nnth (nkfilter f 0 (ndropns σ)) j) σ
using *ndropns-nnth*[*of (nnth (nkfilter f 0 (ndropns σ)) j)]*
using *30* **by** *blast*
show *?thesis* **by** (*simp add*: *1 2 3*)
qed

lemma *pifilt-nnth*:
assumes (∃ *i*≤*nlength σ. f (ndropn i σ)*)
i ≤ nlength (pifilt σ f)
shows (∃ *k ≤ nlength σ. nnth (pifilt σ f) i = nnth σ k*)

using *assms sfxfilt-pifilt-nnth-ndropn*[*of* σ f i]
by (*simp add*: *pifilt-def*)
(*metis enat-the-enat infinity-ileE linorder-le-cases ndropn-all ndropn-nfirst*)

lemma *sfxfilt-nlength-imp*:
assumes ($\exists\, i{\leq}$*nlength* σ. f (*ndropn* i σ) $\wedge$ g (*ndropn* i σ))
shows *nlength* (*sfxfilt* σ (*LIFT*(f $\wedge$ g))) $\leq$ *nlength* (*sfxfilt* σ f)
proof −
have *1*: *nlength* (*sfxfilt* σ (*LIFT*(f $\wedge$ g))) =
nlength (*nfilter* (λ*ys*. f *ys* $\wedge$ g *ys*) (*ndropns* σ))
by (*simp add*: *sfxfilt-def*)
have *2*: *nlength* (*nfilter* f (*ndropns* σ)) = *nlength* (*sfxfilt* σ f)
by (*simp add*: *sfxfilt-def*)
have *3*: $\exists$ $x \in$ *nset* (*ndropns* σ). f x $\wedge$ g x
using *assms in-nset-ndropns* **by** *blast*
have *4*: *nlength* (*nfilter* (λx. f x $\wedge$ g x) (*ndropns* σ)) $\leq$ *nlength* (*nfilter* f (*ndropns* σ))
using *3 nfilter-nlength-imp*[*of ndropns* σ f g] **by** *auto*
show *?thesis* **using** *1 2 4* **by** *auto*
qed

lemma *pifilt-nlength-imp*:
assumes ($\exists\, i{\leq}$*nlength* σ. f (*ndropn* i σ) $\wedge$ g (*ndropn* i σ))
shows *nlength* (*pifilt* σ (*LIFT*(f $\wedge$ g))) $\leq$ *nlength* (*pifilt* σ f)
using *assms*
by (*simp add*: *sfxfilt-nlength-imp pifilt-def*)

lemma *nellist-nset-sfxfilt* [*simp*]:
assumes ($\exists$ $i \leq$ *nlength xs*. f (*ndropn* i *xs*))
shows (*nset* (*sfxfilt xs* f)) = {*ys*. *ys* $\in$ *nset* (*ndropns xs*) $\wedge$ f (*ys*)}
using *assms*
proof −
have *1*: *nset* (*sfxfilt xs* f) = *nset* (*nfilter* f (*ndropns xs*))
by (*simp add*: *sfxfilt-def*)
have *2*: $\exists$ $x \in$ *nset* (*ndropns xs*). f x
using *assms* **using** *in-nset-ndropns* **by** *blast*
have *3*: *nset* (*nfilter* f (*ndropns xs*)) = {*ys*. *ys* $\in$ *nset* (*ndropns xs*) $\wedge$ f *ys*}
using *2 nset-nfilter*[*of ndropns xs* f] **by** *auto*
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *subset-nfilter*:
assumes $\exists\, x \in$ *nset xs*. P x
shows *nset*(*nfilter* P *xs*) $\leq$ *nset*(*nfilter* (λx. P x $\vee$ Q x) *xs*)
proof −
have *1*: $\exists\, x \in$ *nset xs*. P x $\vee$ Q x
using *assms* **by** *blast*
have *2*: *nset*(*nfilter* (λx. P x $\vee$ Q x) *xs*) = {$x \in$ *nset xs*. P x $\vee$ Q x}
using *assms nset-nfilter*[*of xs* (λx. P x $\vee$ Q x)] **by** *blast*
have *3*: *nset*(*nfilter* P *xs*) = {$x{\in}$ *nset xs*. P x}

using *assms nset-nfilter*[*of xs P*] **by** (*simp add*: *Collect-conj-eq*)
have *4*: $\{x \in nset\ xs.\ P\ x\} \leq \{x \in nset\ xs.\ P\ x \vee Q\ x\}$
by *auto*
show *?thesis* **by** (*simp add*: *2 3 4*)
qed

lemma *nellist-subset-sfxfilt* [*simp*]:
assumes $(\exists\ i \leq nlength\ xs.\ f\ (ndropn\ i\ xs))$
shows $(nset\ (sfxfilt\ xs\ f)) \leq (nset\ (sfxfilt\ xs\ (LIFT(f \vee g))))$
proof −
have *1*: $\exists\ x \in nset\ (ndropns\ xs).\ f\ x$
using *assms* **using** *in-nset-ndropns* **by** *blast*
have *2*: $nset\ (sfxfilt\ xs\ f) = nset\ (nfilter\ f\ (ndropns\ xs))$
by (*simp add*: *sfxfilt-def*)
have *3*: $nset\ (sfxfilt\ xs\ (LIFT(f \vee g))) = nset\ (nfilter\ (\lambda x.\ f\ x \vee g\ x)\ (ndropns\ xs))$
by (*simp add*: *sfxfilt-def*)
have *4*: $nset\ (nfilter\ f\ (ndropns\ xs)) \leq nset\ (nfilter\ (\lambda x.\ f\ x \vee g\ x)\ (ndropns\ xs))$
using *1 subset-nfilter*[*of ndropns xs f g*] **by** *auto*
show *?thesis* **using** *2 3 4* **by** *blast*
qed

lemma *nellist-nset-nmap*:
$(nset\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ xs)) = \{(nnth\ ys\ 0) \mid ys.\ ys \in nset\ xs\}$
by (*auto simp add*: *in-nset-conv-nnth nset-conv-nnth*)
(*metis nnth-nmap*)

lemma *nellist-nset-pifilt* [*simp*]:
assumes $(\exists\ i \leq nlength\ xs.\ f\ (ndropn\ i\ xs))$
shows $(nset\ (pifilt\ xs\ f)) = \{(nnth\ ys\ 0) |\ ys.\ ys \in nset(ndropns\ xs) \wedge f\ (ys)\}$
proof −
have *1*: $(nset\ (pifilt\ xs\ f)) = (nset\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ xs\ f)))$
by (*simp add*: *pifilt-def*)
have *2*: $(nset\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ xs\ f))) =$
$\{(nnth\ ys\ 0) \mid ys.\ ys \in nset\ (sfxfilt\ xs\ f)\}$
using *nellist-nset-nmap*[*of sfxfilt xs f*] **by** *auto*
have *3*: $\{(nnth\ ys\ 0) \mid ys.\ ys \in nset\ (sfxfilt\ xs\ f)\} =$
$\{(nnth\ ys\ 0) |\ ys.\ ys \in nset(ndropns\ xs) \wedge f\ (ys)\}$
using *assms* **by** *auto*
show *?thesis* **using** *1 2 3* **by** *auto*
qed

lemma *nellist-nnth-sfxfilt-in-nset*:
$x \in nset\ (sfxfilt\ \sigma\ (LIFT(f \vee g))) =$
$(\exists\ i \leq nlength\ (sfxfilt\ \sigma\ (LIFT(f \vee g))).\ x = (nnth\ (sfxfilt\ \sigma\ (LIFT(f \vee g)))\ i))$
by (*metis in-nset-conv-nnth*)

lemma *nfilter-nnth-or*:
assumes $\exists\ x \in nset\ xs.\ P\ x$
shows $\exists\ x \in nset(nfilter\ (\lambda x.\ P\ x \vee Q\ x)\ xs).\ P\ x$
proof −

have *0*: $\exists\ x \in nset\ xs.\ P\ x \vee Q\ x$
using *assms* **by** *auto*
have *1*: $\exists\, i \leq nlength(nfilter\ (\lambda x.\ P\ x \vee Q\ x)\ xs).\ P\ (nnth\ (nfilter\ P\ xs)\ i)$
using *assms nkfilter-nnth-aa*[*of xs* ($\lambda x.\ P\ x$)] *nkfilter-nnth-aa*
by (*metis* (*mono-tags*, *lifting*) *le-cases le-zero-eq zero-enat-def*)
obtain *i* **where** *2*: $i \leq nlength(nfilter\ (\lambda x.\ P\ x \vee Q\ x)\ xs) \wedge\ P\ (nnth\ (nfilter\ P\ xs)\ i) \wedge$
$(nnth\ (nfilter\ P\ xs)\ i) \in nset\ (nfilter\ P\ xs)$
by (*metis 1 in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *3*: $nset\ (nfilter\ P\ xs) \leq nset(nfilter\ (\lambda x.\ P\ x \vee Q\ x)\ xs)$
using *assms subset-nfilter*[*of xs P Q*] **by** *auto*
have *4*: $(nnth\ (nfilter\ P\ xs)\ i) \in nset\ (nfilter\ P\ xs)$
using *2* **by** *auto*
have *5*: $(nnth\ (nfilter\ P\ xs)\ i) \in nset(nfilter\ (\lambda x.\ P\ x \vee Q\ x)\ xs)$
using *3 4* **by** *blast*
show *?thesis* **using** *2 5* **by** *blast*
qed

lemma *sfxfilt-nnth-or*:
assumes $(\exists\ i \leq nlength\ \sigma.\ (ndropn\ i\ \sigma) \models f)$
shows $(\exists\ i \leq nlength\ (sfxfilt\ \sigma\ (LIFT(f \vee g)))).\ (nnth\ (sfxfilt\ \sigma\ (LIFT(f \vee g))\ )\ i) \models f)$
proof −
have *1*: $\exists\ x \in nset\ (ndropns\ \sigma).\ f\ x$
using *assms* **by** (*simp add*: *sfxfilter-help*)
have *2*: $sfxfilt\ \sigma\ (LIFT(f \vee g)) = nfilter\ (\lambda\ x.\ f\ x \vee g\ x)\ (ndropns\ \sigma)$
by (*simp add*: *sfxfilt-def*)
have *3*: $\exists\ x \in nset(nfilter\ (\lambda\ x.\ f\ x \vee g\ x)\ (ndropns\ \sigma)).\ f\ x$
using *1 nfilter-nnth-or*[*of ndropns* σ *f g*] **by** *auto*
from *2 3* **show** *?thesis*
by (*metis* (*full-types*) *in-nset-conv-nnth*)
qed

lemma *NotPiFalse*:
$\sigma \models \neg\ ((\#False)\ \Pi\ f)$
by (*simp add*: *pi-d-def*)

lemma *pifilt-true*:
$pifilt\ \sigma\ (LIFT(\#True)) = \sigma$
by (*simp add*: *pifilt-def sfxfilt-def nmap-first-ndropns*)

lemma *pifilt-state-help*:
$(\exists\ x \in nset\ (ndropns\ xs)\ .\ (LIFT(init\ w))\ x) = (\exists\ x \in nset\ xs.\ w\ ((NNil\ x)))$
proof (*auto simp add*: *init-defs*)
show $\bigwedge x.\ \ x \in nset\ (ndropns\ xs) \Longrightarrow w\ (NNil\ (nfirst\ x)) \Longrightarrow \exists\, x{\in}nset\ xs.\ w\ (NNil\ x)$
using *in-nset-ndropns* **by** (*metis in-nset-conv-nnth ndropn-nfirst*)
show $\bigwedge x.\ x \in nset\ xs \Longrightarrow w\ (NNil\ x) \Longrightarrow \exists\, x{\in}nset\ (ndropns\ xs).\ w\ (NNil\ (nfirst\ x))$
by (*metis in-nset-ndropns-nhd ndropn-0 ndropn-nfirst*)
qed

lemma *pifilt-init*:
shows $(pifilt\ xs\ (\lambda s.\ s \models init\ w)) = nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ xs$

proof (*simp add*: *init-defs pifilt-def sfxfilt-def*)
show *nmap* (λ*s*. *nnth s 0*) (*nfilter* (λ*s*. *w* (*NNil* (*nfirst s*))) (*ndropns xs*)) =
nfilter (λ*y*. *w* (*NNil y*)) *xs*
proof −
have *1*: ⋀*x*. ((λ*y*. *w* ((*NNil y*))) ∘ (λ*s*. *nnth s 0*)) *x* = (λ*s*. *w* (*NNil* (*nfirst s*))) *x*
by *simp* (*metis ndropn-0 ndropn-nfirst*)
have *2*: ((λ*y*. *w* ((*NNil y*))) ∘ (λ*s*. *nnth s 0*)) = (λ*s*. *w* (*NNil* (*nfirst s*)))
using *1* **by** *blast*
have *4*: *nmap* (λ*s*. *nnth s 0*) (*nfilter* (λ*s*. *w* (*NNil* (*nfirst s*))) (*ndropns xs*)) =
nfilter (λ*y*. *w* ((*NNil y*))) (*nmap* (λ*s*. *nnth s 0*) (*ndropns xs*))
by (*metis* (*mono-tags*, *lifting*) *1* *nfilter-cong nfilter-nmap*)
have *5*: (*nmap* (λ*s*. *nnth s 0*) (*ndropns xs*)) = *xs*
by (*simp add*: *nmap-first-ndropns*)
show *?thesis*
by (*simp add*: *4 5*)
qed
qed

lemma *pifilt-init-a*:
shows (*pifilt xs* (λ*s*. *w* (*NNil* (*nnth s 0*)))) = *nfilter* (λ*y*. *w* (*NNil y*)) *xs*
proof −
have *2*: (*pifilt xs* (λ*s*. *s* ⊨ *init w*)) = (*pifilt xs* (λ*s*. *w* (*NNil* (*nnth s 0*))))
by (*metis* (*no-types*, *lifting*) *in-nset-ndropns init-defs ndropn-nfirst nfilter-cong nnth-zero-ndropn*
ntaken-0 pifilt-def sfxfilt-def)
show *?thesis* **using** *pifilt-init 2* **by** (*metis*)
qed

lemma *pifilt-pifilt* :
assumes (∃ *i*≤*nlength xs*. *f* (*ndropn i xs*))
(∃ *i*≤*nlength* (*pifilt xs f*). *w* (*NNil* (*nnth* (*ndropn i* (*pifilt xs f*)) *0*)))
shows (*pifilt* (*pifilt xs f*) (*LIFT*(*init w*))) = *pifilt xs* (*LIFT*(*f* ∧ *init w*))
proof −
have *1*: ∃ *i*≤*nlength* (*pifilt xs f*). (*LIFT*(*init w*)) (*ndropn i* (*pifilt xs f*))
using *assms* **by** (*simp add*: *init-defs ndropn-nfirst*)
have *2*: (*pifilt* (*pifilt xs f*) (*LIFT*(*init w*))) = *nfilter* (λ*y*. *w* ((*NNil y*))) (*pifilt xs f*)
using *1 pifilt-init*[*of* (*pifilt xs f*) *w*] **by** *auto*
have *3*: (*pifilt xs f*) = *nmap* (λ*s*. *nnth s 0*) (*sfxfilt xs f*)
by (*simp add*: *assms sfxfilt-pifilt*)
have *4*: (*sfxfilt xs f*) = *nfilter* (λ *ys*. *f ys*) (*ndropns xs*)
using *sfxfilt-def* **by** *blast*
have *5*: (*pifilt xs f*) = *nmap* (λ*s*. *nnth s 0*) (*nfilter* (λ *ys*. *f ys*) (*ndropns xs*))
by (*simp add*: *3 4*)
have *6*: *nfilter* (λ*y*. *w* (*NNil y*)) (*pifilt xs f*) =
nfilter (λ*y*. *w* (*NNil y*)) (*nmap* (λ*s*. *nnth s 0*) (*nfilter* (λ *ys*. *f ys*) (*ndropns xs*)))
using *5* **by** *simp*
have *7*: *nfilter* (λ*y*. *w* (*NNil y*)) (*nmap* (λ*s*. *nnth s 0*) (*nfilter* (λ *ys*. *f ys*) (*ndropns xs*))) =
nmap (λ*s*. *nnth s 0*) (*nfilter* ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) (*nfilter* (λ *ys*. *f ys*) (*ndropns xs*)))
using *assms 3 4* **by** (*simp add*: *nfilter-nmap pifiltinit-help*)
have *8*: ∃ *x*∈*nset* (*nfilter* (λ *ys*. *f ys*) (*ndropns xs*)). ((λ*y*. *w* ((*NNil y*)))∘(λ*s*. *nnth s 0*)) *x*
using *assms 3 4* *in-nset-conv-nnth*[*of* - (*nfilter f* (*ndropns xs*))]

by *simp*
(*metis in-nset-conv-nnth ndropn-0 ndropn-nfirst nlength-nmap nnth-nmap*)
have *9*: $\exists\, x{\in}nset\ (ndropns\ xs).\ (\lambda\ ys.\ f\ ys)\ x$
using *assms* **by** (*simp add*: *sfxfilter-help*)
have *10*: $\exists\, x{\in}nset\ (ndropns\ xs).\ ((\lambda y.\ w\ (NNil\ y))\circ(\lambda s.\ nnth\ s\ 0))\ x \wedge (\lambda\ ys.\ f\ ys)\ x$
using *8 9*
using *exist-conj*[*of f* (*ndropns xs*) ((λ*y*. *w* (*NNil y*)) ∘ (λ*s*. *nnth s 0*))] **by** *fastforce*
have *11*: *nfilter* ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) (*nfilter* (λ *ys*. *f ys*) (*ndropns xs*)) =
nfilter (λ *zs*. ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) (*ndropns xs*)
using *nfilter-nfilter*[*of* (λ *ys*. *f ys*) (*ndropns xs*) ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*))]
8 10 **by** *blast*
have *12*: $\exists\, i{\le}nlength\ xs.\ (\lambda\ zs.\ ((\lambda y.\ w\ (NNil\ y))\circ(\lambda s.\ nnth\ s\ 0))\ zs \wedge (\lambda\ ys.\ f\ ys)\ zs)\ (ndropn\ i\ xs)$
by (*metis 10 in-nset-ndropns*)
have *13*: (*nfilter* (λ *zs*. ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) (*ndropns xs*))
= (*sfxfilt xs* (λ *zs*. ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*))
by (*simp add*: *sfxfilt-def*)
have *14*: $\exists\, i{\le}nlength\ xs.\ (\lambda\ zs.\ ((\lambda y.\ w\ (NNil\ y))\circ(\lambda s.\ nnth\ s\ 0))\ zs \wedge (\lambda\ ys.\ f\ ys)\ zs)\ (ndropn\ i\ xs)$
using *12* **by** *blast*
have *15*: *nmap* (λ*s*. *nnth s 0*)
((*sfxfilt xs* (λ *zs*. ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*))) =
pifilt xs (λ *zs*. ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*)
using *14* **by** (*simp add*: *sfxfilt-pifilt*)
have *16*: $\bigwedge xs\ .\ (\lambda\ zs.\ ((\lambda y.\ w\ (NNil\ y))\circ(\lambda s.\ nnth\ s\ 0))\ zs \wedge (\lambda\ ys.\ f\ ys)\ zs)\ xs =$
(*LIFT*(*f* ∧ *init w*)) *xs*
by (*simp add*: *init-defs*) (*metis ndropn-0 ndropn-nfirst*)
have *17*: *pifilt xs* (λ *zs*. ((λ*y*. *w* (*NNil y*))∘(λ*s*. *nnth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) =
pifilt xs (*LIFT*(*f* ∧ *init w*))
using *16* **by** *presburger*
show *?thesis*
using *11 13 15 17 2 3 4 7* **by** *auto*
qed

lemma *PiStatesem*:
(σ ⊨ (*init w*) Π *f*) =
((∃ *i*≤*nlength* σ. *w* (*NNil* (*nnth* σ *i*))) ∧ *f* (*nfilter* (λ*y*. *w* ((*NNil y*))) σ))
by (*simp add*: *pi-d-def init-defs ndropn-nfirst pifilt-init*)

3.9.3 Soundness of Axioms

PiK

lemma *PiKsem*:
$\sigma \models f1\ \Pi^u\ (\ f \longrightarrow g) \longrightarrow (\ \ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ )$
by (*simp add*: *upi-d-def init-defs pi-d-def*) *auto*

lemma *PiK*:
$\vdash f1\ \Pi^u\ (f \longrightarrow g) \longrightarrow (\ \ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ )$
using *PiKsem Valid-def* **by** *blast*

PiDc

lemma *PiDcsem*:
$\sigma \models f\ \Pi\ g \longrightarrow f\ \Pi^u\ g$
by (*simp add*: *upi-d-def init-defs pi-d-def*)

lemma *PiDc*:
$\vdash f\ \Pi\ g \longrightarrow f\ \Pi^u\ g$
using *PiDcsem Valid-def* **by** *blast*

PiN

lemma *PiN*:
assumes $\vdash g$
shows $\vdash f\ \Pi^u\ g$
using *assms* **by** (*simp add*: *Valid-def pi-d-def upi-d-def*)

PiTrueEqvDiamond

lemma *PiTrueEqvDiamond*:
$\vdash f\ \Pi\ \#True = \Diamond\ f$
by (*simp add*: *Valid-def pi-d-def itl-defs*)

PiOr

lemma *PiOr*:
$\vdash f\ \Pi\ (g1 \vee g2) = (f\ \Pi\ g1 \vee f\ \Pi\ g2)$
by (*simp add*: *Valid-def pi-d-def*) *blast*

UPiFalseEqvBoxNot:

lemma *UPiFalseEqvBoxNot*:
$\vdash f\ \Pi^u\ \#False = \Box\ (\neg f)$
by (*simp add*: *Valid-def upi-d-def pi-d-def itl-defs*)

BoxEqvImpPiEqv

lemma *BoxEqvImpPiEqvsem*:
assumes $(\sigma \models \Box\ (f1 = f2))$
shows $(\sigma \models (f1\ \Pi\ g = f2\ \Pi\ g))$
proof −
show $(\sigma \models (f1\ \Pi\ g = f2\ \Pi\ g))$
proof −
have *1*: $\forall\, n{\leq}nlength\ \sigma.\ f1\ (ndropn\ n\ \sigma) = f2\ (ndropn\ n\ \sigma)$
using *assms* **by** (*simp add*: *itl-defs*)
have *2*: $(\sigma \models (f1\ \Pi\ g)) = ((\exists\, i{\leq}nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f1))$
by (*simp add*: *pi-d-def*)
have *3*: $(\exists\, i{\leq}nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) = (\exists\, i{\leq}nlength\ \sigma.\ f2\ (ndropn\ i\ \sigma))$
using *1* **by** *blast*
have *6*: $(\exists\, i{\leq}nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \Longrightarrow$
$\{\ (ndropn\ n\ \sigma)\ |\ n.\ n \leq nlength\ \sigma \wedge f1\ (ndropn\ n\ \sigma)\} =$
$\{\ (ndropn\ n\ \sigma)\ |\ n.\ n \leq nlength\ \sigma \wedge f2\ (ndropn\ n\ \sigma)\}$

using *1* **by** *blast*
have *7*: $(\exists\, i{\leq}nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \Longrightarrow (sfxfilt\ \sigma\ f1) = (sfxfilt\ \sigma\ f2)$
by (*metis 1 in-nset-ndropns nfilter-cong sfxfilt-def*)
have *8*: $((\exists\, i{\leq}nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f1)) =$
$((\exists\, i{\leq}nlength\ \sigma.\ f2\ (ndropn\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f2))$
by (*metis 3 7 pifilt-def*)
show *?thesis*
by (*simp add*: *8 pi-d-def*)
qed
qed

lemma *BoxEqvImpPiEqv*:
$\vdash \Box\ (f1 = f2) \longrightarrow (f1\ \Pi\ g = f2\ \Pi\ g)$
using *BoxEqvImpPiEqvsem* **by** (*simp add*: *Valid-def*,*auto*)

PiDiamondImpDiamond

lemma *PiDiamondImpDiamondsem*:
$\sigma \models f\ \Pi\ (\Diamond\ (init\ w)) \longrightarrow \Diamond\ (init\ w)$
by (*simp add*: *Valid-def pi-d-def itl-defs ndropn-nfirst*)
(*metis pifilt-nnth*)

lemma *PiDiamondImp*:
$\vdash f\ \Pi\ (\Diamond\ (init\ w)) \longrightarrow \Diamond\ (init\ w)$
using *PiDiamondImpDiamondsem Valid-def* **by** *blast*

PiAssoc

lemma *PiAssocsem1*:
assumes $i \leq nlength\ \sigma$
$f\ (ndropn\ i\ \sigma)$
$ia \leq nlength\ (pifilt\ \sigma\ f)$
$w\ (NNil\ (nnth\ (pifilt\ \sigma\ f)\ ia))$
shows $\exists\, i{\leq}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ \sigma)\ 0))$
proof −
have *1*: $(nnth\ (pifilt\ \sigma\ f)\ ia) = (nnth\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia)$
using *assms(1) assms(2) assms(3) sfxfilt-pifilt-nnth* **by** *blast*
have *2*: $(nnth\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia) =$
$(\lambda s.\ nnth\ s\ 0)\ (nnth\ (sfxfilt\ \sigma\ f)\ ia)$
by (*metis assms(3) nlength-nmap nnth-nmap pifilt-def*)
have *3*: $f\ (nnth\ (sfxfilt\ \sigma\ f)\ ia)$
using *sfxfilt-nnth*
by (*metis assms(1) assms(2) assms(3) nlength-nmap pifilt-def*)
have *4*: $nnth\ (sfxfilt\ \sigma\ f)\ ia = ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ia)\ \sigma$
using *sfxfilt-pifilt-nnth-ndropn assms(1) assms(2) assms(3)* **by** *blast*
have *5*: $w\ (NNil\ (nnth\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ia)\ \sigma)\ \ 0))$
using *1 2 4 assms(4)* **by** *auto*
show *?thesis* **by** (*metis 3 4 5 enat-ile le-cases ndropn-all*)
qed

lemma *PiAssocsem2*:

assumes $i \leq nlength\ \sigma$
$f\ (ndropn\ i\ \sigma)$
$w\ (NNil\ (nnth\ \sigma\ i))$
shows $\exists j{\leq}nlength\ (pifilt\ \sigma\ f).\ w\ (NNil\ (nnth\ (pifilt\ \sigma\ f)\ j))$
proof −
have *1*: $\exists j{\leq}nlength\ (sfxfilt\ \sigma\ f).\ f\ (nnth\ (sfxfilt\ \sigma\ f)\ j)$
using *assms pifilt-exists* **by** *blast*
have *2*: $(LIFT\ (init\ w))\ (ndropn\ i\ \sigma)$
by (*simp add*: *assms init-defs ndropn-nfirst*)
have *3*: $\exists j{\leq}nlength\ (sfxfilt\ \sigma\ (LIFT(init\ w))).\ (LIFT(init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT(init\ w)))\ j)$
using *pifilt-exists 2 assms* **by** *blast*
have *4*: $(LIFT\ (f \wedge init\ w))\ (ndropn\ i\ \sigma)$
by (*simp add*: *2 assms*)
have *5*: $\exists j{\leq}nlength\ (sfxfilt\ \sigma\ (LIFT(f \wedge init\ w))).$
$(LIFT(f \wedge init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT(f \wedge init\ w)))\ j)$
using *pifilt-exists 4 assms* **by** *blast*
have *6*: $\exists i{\leq}nlength\ \sigma.\ ndropn\ i\ \sigma \models f \wedge init\ w$
using *4 assms* **by** *blast*
have *7*: $\exists j{\leq}nlength\ (sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg\ (init\ w)))\ \ )).$
$(LIFT(f \wedge init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg(init\ w)))\ ))\ j)$
using *6 sfxfilt-nnth-or*[*of* σ $LIFT(f \wedge init\ w)$ $LIFT(f \wedge \neg(init\ w))$]
by *auto*
have *8*: $\bigwedge\sigma\ .\ (\sigma \models ((f \wedge init\ w) \vee (f \wedge \neg\ (init\ w)))) = (\sigma \models f)$
by *auto*
have *9*: $(sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg\ (init\ w)))\ \ )) =$
$(sfxfilt\ \sigma\ f)$
using *8* **by** (*simp add*: *sfxfilt-def*)
have *10*: $\exists j{\leq}nlength\ (sfxfilt\ \sigma\ f).$
$(LIFT(f \wedge init\ w))\ (nnth\ (sfxfilt\ \sigma\ f)\ j)$
using *7 9* **by** *auto*
have *11*: $nlength\ (sfxfilt\ \sigma\ f) = nlength\ (pifilt\ \sigma\ f)$
by (*simp add*: *pifilt-def*)
have *12*: $\exists j{\leq}nlength\ (pifilt\ \sigma\ f).$
$(LIFT(init\ w))\ (nnth\ (sfxfilt\ \sigma\ f)\ j)$
using *10 11* **by** *auto*
from *12 11* **show** *?thesis* **unfolding** *pifilt-def init-defs*
by (*metis nlast-NNil nnth-nmap ntaken-0 ntaken-nlast*)
qed

lemma *PiAssocsema*:
$((\exists i{\leq}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)) \wedge$
$(\exists i{\leq}nlength\ (pifilt\ \sigma\ f).\ w\ (NNil\ (nnth\ (ndropn\ i\ (pifilt\ \sigma\ f))\ 0))\ )) =$
$(\exists i{\leq}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ \sigma)\ 0)))$
using *PiAssocsem1*[*of* - σ *f* - *w*] *PiAssocsem2*[*of* - σ *f w*] **by** *fastforce*

lemma *PiAssocsemb*:
$((\exists i{\leq}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)) \wedge$
$(\exists i{\leq}nlength\ (pifilt\ \sigma\ f).\ (LIFT(init\ w))\ (ndropn\ i\ (pifilt\ \sigma\ f))\ )) =$
$(\exists i{\leq}nlength\ \sigma.\ (LIFT(f \wedge init\ w))\ (ndropn\ i\ \sigma))$
using *PiAssocsem1*[*of* - σ *f* - *w*] *PiAssocsem2*[*of* - σ *f w*]

by (*simp add*: *init-defs ndropn-nfirst*) *blast*

lemma *PiAssocsem*:
$\sigma \models f\ \Pi\ ((init\ w)\ \Pi\ \ g) = \quad (f \wedge (init\ w))\ \Pi\ g$
proof (*auto simp add*: *pi-d-def init-defs*)
fix *i*
fix *ia*
assume *a0*: $g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$
assume *a1*: $(enat\ i) \leq nlength\ \sigma$
assume *a2*: $f\ (ndropn\ i\ \sigma)$
assume *a3*: $(enat\ ia) \leq nlength\ (pifilt\ \sigma\ f)$
assume *a4*: $w\ (NNil\ (nfirst\ (ndropn\ ia\ (pifilt\ \sigma\ f))))$
show $\exists\, i.\ enat\ i \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma) \wedge w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$
using *a0 a1 a2 a3 a4 PiAssocsem1*
by (*metis ndropn-nfirst nnth-zero-ndropn*)
next
fix *i*
fix *ia*
assume *a0*: $g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$
assume *a1*: $(enat\ i) \leq nlength\ \sigma$
assume *a2*: $f\ (ndropn\ i\ \sigma)$
assume *a3*: $(enat\ ia) \leq nlength\ (pifilt\ \sigma\ f)$
assume *a4*: $w\ (NNil\ (nfirst\ (ndropn\ ia\ (pifilt\ \sigma\ f))))$
show $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$
using *a0 a1 a2 a3 a4* **by** (*metis ndropn-nfirst nnth-zero-ndropn pifilt-pifilt*)
next
fix *i*
assume *a0*: $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$
assume *a1*: $(enat\ i) \leq nlength\ \sigma$
assume *a2*: $f\ (ndropn\ i\ \sigma)$
assume *a3*: $w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$
show $\exists\, i.\ enat\ i \leq nlength\ (pifilt\ \sigma\ f) \wedge w\ (NNil\ (nfirst\ (ndropn\ i\ (pifilt\ \sigma\ f))))$
using *a0 a1 a2 a3* **by** (*metis PiAssocsem2 ndropn-nfirst*)
next
fix *i*
assume *a0*: $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$
assume *a1*: $(enat\ i) \leq nlength\ \sigma$
assume *a2*: $f\ (ndropn\ i\ \sigma)$
assume *a3*: $w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$
show $g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$
using *a0 a1 a2 a3* **by** (*metis PiAssocsem2 ndropn-nfirst nnth-zero-ndropn pifilt-pifilt*)
qed

lemma *PiAssoc*:
$\vdash f\ \Pi\ ((init\ w)\ \Pi\ \ g) = \quad (f \wedge (init\ w))\ \Pi\ g$
using *PiAssocsem Valid-def* **by** *blast*

PiNotEqvDiamondAndNotPi

lemma *PiNotEqvDiamondAndNotPisem*:
$\sigma \models f\ \Pi\ (\neg\ g) = (\Diamond f \wedge \neg(f\ \Pi\ g))$
by (*simp add*: *pi-d-def itl-defs*) *blast*

lemma *PiNotEqvDiamondAndNotPi*:
$\vdash f\ \Pi\ (\neg\ g) = (\Diamond f \wedge \neg(f\ \Pi\ g))$
using *PiNotEqvDiamondAndNotPisem Valid-def* **by** *blast*

PiSChopDist

lemma *PiSChopDistsema*:
assumes $(\sigma \models (init\ w)\ \Pi\ (g \frown h))$
shows $(\sigma \models ((init\ w)\ \Pi\ g) \frown ((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
proof −
have *1*: $(\sigma \models (init\ w)\ \Pi\ (g \frown h))$
using *assms* **by** *auto*
have *2*: $((\exists\, i{\leq}nlength\ \sigma.\ (LIFT(init\ w))\ (ndropn\ i\ \sigma)) \wedge$
$((pifilt\ \sigma\ (LIFT(init\ w))) \models g \frown h)$
$)$
using *1* **by** (*simp add*: *pi-d-def*)
have *3*: $(\exists\, i{\leq}nlength\ \sigma.\ (LIFT(init\ w))\ (ndropn\ i\ \sigma))$
using *2* **by** *auto*
have *4*: $((pifilt\ \sigma\ (LIFT(init\ w))) \models g \frown h)$
using *2* **by** *auto*
have *5*: $nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ \sigma \models g \frown h$
using *pifilt-init* **by** (*metis 4*)
have *6*: $(\exists\, n.\ (enat\ n){\leq}nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \wedge$
$g\ (ntaken\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)) \wedge$
$h\ (ndropn\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)))$
using *5* **by** (*simp add*: *itl-defs*)
obtain *n* **where** *7*: $(enat\ n){\leq}nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \wedge$
$g\ (ntaken\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)) \wedge$
$h\ (ndropn\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma))$
using *6* **by** *auto*
have *8*: $\exists\, i{\leq}nlength\ \sigma.\ w\ (NNil\ (nnth\ \sigma\ i))$
using *3* **using** *init-defs PiStatesem assms* **by** *blast*
have *9*: $\exists\ x \in nset\ \sigma.\ w\ (NNil\ x)$
using *8* **by** (*simp add*: *pifiltinit-help*)
have *10*: $(ntaken\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)) =$
$(nfilter\ (\lambda y.\ w\ (NNil\ y))\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y.\ w\ (NNil\ y))\ 0\ \sigma)\ n)\ )\ \sigma)\ )$
by (*simp add*: *7 9 nfilter-nkfilter-ntaken-1 nkfilter-nlength*)
have *11*: $(ndropn\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)) =$
$(nfilter\ (\lambda y.\ w\ (NNil\ y))\ (ndropn\ ((nnth\ (nkfilter\ (\lambda y.\ w\ (NNil\ y))\ 0\ \sigma)\ n)\ )\ \sigma)\ )$
by (*simp add*: *7 9 nfilter-nkfilter-ndropn-1 nkfilter-nlength*)
have *12*: $g\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y.\ w\ (NNil\ y))\ 0\ \sigma)\ n)\ )\ \sigma)\ )$
using *10 7* **by** *auto*
have *13*: $h\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (ndropn\ ((nnth\ (nkfilter\ (\lambda y.\ w\ (NNil\ y))\ 0\ \sigma\ )\ n)\ )\ \sigma)\ )$
using *11 7* **by** *auto*
have *14*: $((nnth\ (nkfilter\ (\lambda y.\ w\ (NNil\ y))\ 0\ \sigma)\ n)\ ) \leq nlength\ \sigma$

using *7 9 nkfilter-upperbound*[*of* σ ($\lambda y.$ w (*NNil* y)) - *0*]
by (*metis gen-nlength-def nkfilter-nlength nlength-code*)
have *15*: w (*NNil* (*nnth* (*ndropn* ((*nnth* (*nkfilter* ($\lambda y.$ w (*NNil* y)) *0* σ) n)) σ) *0*))
using *7 9 nkfilter-nmap-nfilter*[*of* σ $\lambda x.$ w (*NNil* x)] *nkfilter-nnth-aa*[*of* σ $\lambda x.$ w (*NNil* x)]
by *simp*
(*metis* (*mono-tags*, *lifting*) *7* *nkfilter-nlength nnth-nmap*)
have *16*: ($\exists$ i. (*enat* i)$\leq$*nlength* (*ntaken* ((*nnth* (*nkfilter* ($\lambda y.$ w (*NNil* y)) *0* σ) n)) σ) $\wedge$
w (*NNil* (*nnth* (*ndropn* i (*ntaken* ((*nnth* (*nkfilter* ($\lambda y.$ w (*NNil* y)) *0* σ) n)) σ)) *0*)))
using *14 15*
by (*metis le-cases min.orderE ndropn-0 ndropn-nfirst ntaken-nlength ntaken-nnth*)
have *17*: ($\exists$ i. (*enat* i)$\leq$*nlength* (*ndropn* ((*nnth* (*nkfilter* ($\lambda y.$ w (*NNil* y)) *0* σ) n)) σ) $\wedge$
w (*NNil* (*nnth* (*ndropn* (i + ((*nnth* (*nkfilter* ($\lambda y.$ w (*NNil* y)) *0* σ) n))) σ) *0*)))
using *15* **by** (*metis add.left-neutral zero-enat-def zero-le*)
have *18*: ($\exists$ n. (*enat* n)$\leq$*nlength* σ $\wedge$
($\exists$ i. (*enat* i) $\leq$*nlength* (*ntaken* n σ) $\wedge$ w (*NNil* (*nnth* (*ndropn* i (*ntaken* n σ)) *0*)) $\wedge$
g (*nfilter* ($\lambda y.$ w ((*NNil* y))) (*ntaken* n σ)) $\wedge$
w (*NNil* (*nnth* (*ndropn* n σ) *0*)) $\wedge$
($\exists$ i. (*enat* i) $\leq$*nlength* (*ndropn* n σ) $\wedge$ w (*NNil* (*nnth* (*ndropn* (i + n) σ) *0*))) $\wedge$
h (*nfilter* ($\lambda y.$ w ((*NNil* y))) (*ndropn* n σ))))
using *12 13 14 15 16 17* **by** *blast*
have *181*: ($\exists$ n. (*enat* n)$\leq$*nlength* σ $\wedge$ ($\exists$ i. (*enat* i)$\leq$ *nlength* (*ntaken* n σ) $\wedge$ w (*NNil* (*nnth* (*ntaken* n σ) i))))
using *18* **by** *auto*
have *182*: ($\exists$ n. (*enat* n)$\leq$*nlength* σ $\wedge$ ($\exists$ i. (*enat* i)$\leq$ *nlength* (*ndropn* n σ) $\wedge$ w (*NNil* (*nnth* σ (i + n)))))
using *18* **by** *auto*
have *19*: ($\exists$ n. (*enat* n)$\leq$*nlength* σ $\wedge$
($\exists$ i. (*enat* i)$\leq$*nlength* (*ntaken* n σ) $\wedge$ w (*NNil* (*nnth* (*ndropn* i (*ntaken* n σ)) *0*)) $\wedge$
g (*pifilt* (*ntaken* n σ) (*LIFT*(*init* w))) $\wedge$
w (*NNil* (*nnth* (*ndropn* n σ) *0*)) $\wedge$
($\exists$ i. (*enat* i)$\leq$*nlength* (*ndropn* n σ) $\wedge$ w (*NNil* (*nnth* (*ndropn* (i + n) σ) *0*)) $\wedge$
h (*pifilt* (*ndropn* n σ) (*LIFT*(*init* w))))))
using *18 pifilt-init*[*of* - w] **by** *auto*
have *20*: (($\exists$ n. (*enat* n)$\leq$*nlength* σ $\wedge$
($\exists$ i. (*enat* i)$\leq$*nlength* (*ntaken* n σ) $\wedge$ (*LIFT*(*init* w)) (*ndropn* i (*ntaken* n σ))) $\wedge$
g (*pifilt* (*ntaken* n σ) (*LIFT*(*init* w))) $\wedge$
(*LIFT*(*init* w)) (*ndropn* n σ) $\wedge$
($\exists$ i. (*enat* i)$\leq$*nlength* (*ndropn* n σ) $\wedge$ (*LIFT*(*init* w)) (*ndropn* (i + n) σ))
$\wedge$ h (*pifilt* (*ndropn* n σ) (*LIFT*(*init* w)))))
using *19*
by (*metis init-defs ndropn-nfirst nnth-zero-ndropn ntaken-0*)
have *21*: (σ $\models$ (($init$ w) Π g)$^\frown$(($init$ w) $\wedge$ (($init$ w) Π h)))
using *20* **by** (*simp add*: *add.commute itl-defs ndropn-ndropn pi-d-def*)
show *?thesis* **by** (*simp add*: *21*)
qed

lemma *PiSChopDistsemb*:
assumes (σ $\models$ (($init$ w) Π g)$^\frown$(($init$ w) $\wedge$ (($init$ w) Π h)))
shows (σ $\models$ ($init$ w) Π ($g^\frown h$))
proof −

have *1*: $(\sigma \models ((init\ w)\ \Pi\ g) \frown ((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
using *assms* **by** *auto*
have *2*: $(\exists n.\ (enat\ n) \leq nlength\ \sigma \wedge$
$((ntaken\ n\ \sigma) \models ((init\ w)\ \Pi\ g)) \wedge$
$((ndropn\ n\ \sigma) \models ((init\ w) \wedge ((init\ w)\ \Pi\ h))))$
using *assms schop-defs* **by** *blast*
obtain *n* **where** *3*: $(enat\ n) \leq nlength\ \sigma \wedge ((ntaken\ n\ \sigma) \models ((init\ w)\ \Pi\ g)) \wedge$
$((ndropn\ n\ \sigma) \models ((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
using *2* **by** *auto*
have *4*: $((\exists i.\ (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma))) \wedge$
$((pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w))) \models g)$
$)$
by (*meson 3 pi-d-def*)
have *5*: $(\exists i.\ (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma)))$
using *4* **by** *auto*
have *6*: $g\ (pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w)))$
using *4* **by** *auto*
have *7*: $g\ (nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (ntaken\ n\ \sigma))$
using *5 6 pifilt-init* **by** (*metis*)
have *8*: $((\exists i.\ (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ndropn\ n\ \sigma))) \wedge$
$((pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w))) \models h)$
$)$
by (*metis 3 intensional-rews(3) pi-d-def*)
have *9*: $(\exists i.\ (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ndropn\ n\ \sigma)))$
using *8* **by** *auto*
have *10*: $h\ (pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w)))$
using *8* **by** *auto*
have *11*: $h\ (nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (ndropn\ n\ \sigma))$
using *10 9 pifilt-init* **by** *metis*
have *12*: $(\lambda y.\ w\ ((NNil\ y)))\ (nlast\ (ntaken\ n\ \sigma))$
by (*metis 3 init-defs intensional-rews(3) ndropn-nfirst ntaken-0 ntaken-nlast*)
have *13*: $\exists\ x \in nset\ \sigma.\ (\lambda y.\ w\ ((NNil\ y)))\ x$
using *12 3* **by** (*metis in-nset-conv-nnth ntaken-nlast*)
have *14*: $\exists\ x \in nset\ (ntaken\ n\ \sigma)\ .\ (\lambda y.\ w\ ((NNil\ y)))\ x$
using *12* **using** *nfinite-ntaken nset-nlast* **by** *blast*
have *15*: $\exists\ x \in nset\ (ndropn\ n\ \sigma)\ .\ (\lambda y.\ w\ ((NNil\ y)))\ x$
by (*metis 12 3 dual-order.order-iff-strict ndropn-Suc-conv-ndropn ndropn-nlast*
nellist.set-intros(1) nellist.set-intros(2) nfinite-ntaken ntaken-all ntaken-nlast the-enat.simps)
have *16*: $(nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (ntaken\ n\ \sigma)) =$
$ntaken\ (the\text{-}enat\ (nlength\ (nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (ntaken\ n\ \sigma))))$
$(nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ \sigma)$
using *12 13 14 15 3 nfilter-chop1-ntaken*[*of n σ* (*λy. w* (*NNil y*))] **by** *auto*
have *17*: $(nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (ndropn\ n\ \sigma)) =$
$ndropn\ (the\text{-}enat\ (nlength\ (nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (ntaken\ n\ \sigma))))$
$(nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ \sigma)$
using *12 13 14 15 3 nfilter-chop1-ndropn*[*of n σ* (*λy. w* (*NNil y*))] **by** *auto*
have *18*: $\exists n.\ (enat\ n) \leq nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \wedge$
$g\ (ntaken\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)) \wedge$
$h\ (ndropn\ n\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma))$
by (*metis 11 16 17 7 enat-the-enat infinity-ileE le-cases ntaken-all*)

have *19*: *nfilter* ($\lambda y.\ w\ ((NNil\ y))$) $\sigma \models g \frown h$
by (*simp add*: *18 schop-defs*)
have *20*: ((*pifilt* σ ($LIFT(init\ w)$)) $\models g \frown h$)
by (*metis 19 pifilt-init*)
have *21*: ($\exists\, i.\ (enat\ i) \leq nlength\ \sigma \wedge (LIFT(init\ w))\ (ndropn\ i\ \sigma)$)
using *3* **by** *auto*
show *?thesis* **by** (*simp add*: *20 21 pi-d-def*)
qed

lemma *PiSChopDistsem*:
$\sigma \models (init\ w)\ \Pi\ (g \frown h) = ((init\ w)\ \Pi\ g) \frown ((init\ w) \wedge ((init\ w)\ \Pi\ h))$
using *PiSChopDistsema PiSChopDistsemb unl-lift2* **by** *blast*

lemma *PiSChopDist*:
$\vdash (init\ w)\ \Pi\ (g \frown h) = ((init\ w)\ \Pi\ g) \frown ((init\ w) \wedge ((init\ w)\ \Pi\ h))$
using *PiSChopDistsem Valid-def* **by** *blast*

PiProp

lemma *Pistate*:
($\sigma \models (init\ w)\ \Pi\ f$) =
$((\exists\ x \in nset\ \sigma.\ w\ (NNil\ x)) \wedge\ (\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \models f)\ )$
proof −
have *1*: ($\sigma \models (init\ w)\ \Pi\ f$) =
$(\ (\exists\ i \leq nlength\ \sigma.\ \ w\ (NNil\ (nnth\ \sigma\ i))) \wedge ((pifilt\ \sigma\ (LIFT(init\ w))) \models f)\ )$
by (*auto simp add*: *pi-d-def init-defs ndropn-nfirst*)
have *2*: ($\exists\, i \leq nlength\ \sigma.\ (LIFT(init\ w))\ (ndropn\ i\ \sigma)$) =
$(\exists\ i \leq nlength\ \sigma.\ \ w\ (NNil\ (nnth\ \sigma\ i)))$
by (*auto simp add*: *init-defs ndropn-nfirst*)
have *3*: ($\exists\ i \leq nlength\ \sigma.\ \ w\ (NNil\ (nnth\ \sigma\ i))) \longrightarrow$
$(pifilt\ \sigma\ (LIFT(init\ w))) = (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)$
using *pifilt-init* **using** *2* **by** *blast*
have *4*: ($\exists\ \ i.\ (enat\ i) \leq nlength\ \sigma \wedge\ \ w\ (NNil\ (nnth\ \sigma\ i))) = (\exists\ x \in nset\ \sigma.\ w\ (NNil\ x))$
using *in-nset-conv-nnth* **by** *force*
show *?thesis*
using *1 3 4* **by** *auto*
qed

lemma *PiPropsem1a*:
($\sigma \models f\ \Pi\ \$p$) =
$((\exists\, i.\ (enat\ i) \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0))\ )$
using *sfxfilt-pifilt-nnth-ndropn*[*of* σ *f*]
by (*simp add*: *pi-d-def current-val-d-def pifilt-def*)
(*metis ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-order*(*1*))

lemma *PiPropsem2a*:
$(\sigma \models (\neg f)\ \mathcal{U}\ (f \wedge \$p)) =$
$(\exists\, k \leq nlength\ \sigma.\ f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall\, j < k.\ \neg f\ (ndropn\ j\ \sigma)))$
by (*simp add*: *until-d-def current-val-d-def ndropn-nfirst*)

lemma *PiPropsem3a*:
assumes $(\sigma \models f\ \Pi\ \$p)$
shows $(\sigma \models (\neg f)\ \mathcal{U}\ (f \wedge \$p))$
proof −
have *1*: $((\exists i.\ (enat\ i) \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge$
$p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ )\ )$
using *assms PiPropsem1a* **by** *auto*
have *2*: $\exists\ x \in nset(ndropns\ \sigma).\ f\ x$
using *1* **by** (*simp add*: *sfxfilter-help*)
have *3*: $\forall\ x \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma)).\ f\ (nnth\ (ndropns\ \sigma)\ x)$
using *2 nkfilter-holds* **by** *fastforce*
have *31*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \leq nlength\ \sigma$
by (*metis 1 2 dual-order.strict-trans2 enat-ord-simps*(*2*) *leI ndropns-nnth nkfilter-not-before*)
have *4*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma)$
by (*metis 3 31 in-nset-ndropns in-nset-ndropns-nhd ndropn-0 ndropns-nnth zero-enat-def zero-le*)
have *5*: $(\forall j{<}\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0).\ \neg f\ (ndropn\ j\ \sigma))$
using *nkfilter-not-before*[*of ndropns* σ *f*] *2*
by (*simp add*: *31 less-imp-le ndropns-nnth order-less-le-subst2*)
have *6*: $(\exists k{\leq}nlength\ \sigma.\ f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall j{<}k.\ \neg f\ (ndropn\ j\ \sigma)))$
using *1 31 4 5* **by** *blast*
show *?thesis* **using** *6 PiPropsem2a* **by** *metis*
qed

lemma *PiPropsem3b*:
assumes $(\sigma \models (\neg f)\ \mathcal{U}\ (f \wedge \$p))$
shows $(\sigma \models f\ \Pi\ \$p)$
proof −
have *1*: $(\exists k{\leq}nlength\ \sigma.\ f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall j{<}k.\ \neg f\ (ndropn\ j\ \sigma)))$
using *assms PiPropsem2a* **by** *auto*
obtain *k* **where** *2*: $k{\leq}nlength\ \sigma \wedge f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall j{<}k.\ \neg f\ (ndropn\ j\ \sigma))$
using *1* **by** *auto*
have *3*: $(\exists i.\ (enat\ i) \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma))$
using *2* **by** *blast*
have *31*: $\exists x \in nset(ndropns\ \sigma).\ f\ x$
using *2 in-nset-ndropns* **by** *auto*
have *331*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \leq nlength\ \sigma$
by (*metis 31 gen-nlength-def ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def zero-le*)
have *32*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma)$
using *nkfilter-holds*[*of ndropns* σ *f 0*] *nkfilter-not-before*[*of ndropns* σ *f*]
by (*metis 2 31 ndropns-nfilter-nnth sfxfilt-def sfxfilt-pifilt-nnth-ndropn zero-enat-def zero-le*)
have *4*: $p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ )$
by (*metis 2 31 32 linorder-neqE-nat ndropns-nnth nkfilter-not-before*)
show *?thesis* **using** *4 3* **by** (*simp add*: *PiPropsem1a*)
qed

lemma *PiPropsema*:
$\sigma \models f\ \Pi\ \$p = (\neg f)\ \mathcal{U}\ (f \wedge \$p)$
using *PiPropsem3a PiPropsem3b unl-lift2* **by** *blast*

lemma *PiProp*:

$\vdash f\ \Pi\ \$p = (\neg\ f)\ \mathcal{U}\ (f \wedge \$p)$
using *PiPropsema Valid-def* **by** *blast*

PiNext

lemma *PiNextsem1*:
$(\sigma \models f\ \Pi\ (\bigcirc\ g)) =$
$((\exists\, i{\leq}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)) \wedge$
$0 < nlength\ (nfilter\ f\ (ndropns\ \sigma)) \wedge$
$g\ (ndropn\ (Suc\ 0)\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
using *zero-enat-def* **by** (*simp add*: *pi-d-def itl-defs pifilt-def sfxfilt-def*)

lemma *PiNextsem2*:
$(\sigma \models (\neg\ f)\ \mathcal{U}\ (f \wedge \bigcirc(f\ \Pi\ g))) =$
$(\exists\, k{\leq}nlength\ \sigma.$
$f\ (ndropn\ k\ \sigma) \wedge$
$k < nlength\ \sigma \wedge$
$(\exists\, i{\leq}nlength\ \sigma - Suc\ k.\ f\ (ndropn\ (Suc\ (i + k))\ \sigma)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ (Suc\ k)\ \sigma)))) \wedge (\forall\, j{<}k.\ \neg\ f\ (ndropn\ j\ \sigma)))$
by (*simp add*: *until-d-def itl-defs pi-d-def pifilt-def sfxfilt-def ndropn-ndropn add.commute*)
(*metis add.right-neutral antisym-conv2 enat.simps*(*3*) *enat-add-sub-same less-eqE zero-enat-def*)

lemma *PiNextsem3*:
assumes $(\sigma \models f\ \Pi\ (\bigcirc\ g))$
shows $(\sigma \models (\neg\ f)\ \mathcal{U}\ (f \wedge \bigcirc(f\ \Pi\ g)))$
proof −
have *1*: $((\exists\, i{\leq}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)) \wedge 0 < nlength\ (nfilter\ f\ (ndropns\ \sigma)) \wedge$
$g\ (ndropn\ (Suc\ 0)\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
using *assms* **by** (*simp add*: *PiNextsem1*)
have *2*: $\exists\ x \in nset(ndropns\ \sigma).\ f\ x$
using *1* **by** (*simp add*: *sfxfilter-help*)
have *3*: $\forall\ x \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma)).\ f\ (nnth\ (ndropns\ \sigma)\ x)$
using *2 nkfilter-holds* **by** *fastforce*
have *31*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \leq nlength\ \sigma$
by (*metis 2 gen-nlength-def i0-lb ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def*)
have *4*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \ \sigma)$
by (*metis 3 31 i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def*)
have *41*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma))$
by (*metis 1 2 One-nat-def eSuc-enat ileI1 in-nset-conv-nnth nkfilter-nlength zero-enat-def*)
have *42*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) \leq nlength\ (ndropns\ \sigma)$
by (*metis 2 41 gen-nlength-def in-nset-conv-nnth nkfilter-upperbound nlength-code*)
have *5*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1)\ \ \sigma)$
using *3 41 42* **by** (*metis ndropns-nlength ndropns-nnth*)
have *6*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) < (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1)$
by (*metis 1 2 One-nat-def ileI1 nidx-nkfilter-expand nkfilter-nlength one-eSuc one-enat-def*)
have *7*: $Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \leq nlength\ \sigma$
by (*metis 42 6 Suc-leI dual-order.trans enat-ord-simps*(*1*) *ndropns-nlength*)
have *8*: $0 < nlength\ (nkfilter\ f\ 0\ (ndropns\ \sigma))$
using *1 2 nkfilter-nlength* **by** *fastforce*
have *9*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) \leq nlength\ \sigma$

by (*metis 42 ndropns-nlength*)
have *10*: (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*)) ≤ (*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*)
using *6 Suc-leI* **by** *blast*
have *11*: (*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*) =
(*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*) − (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*)) +
(*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*))
using *10* **by** *auto*
have *12*: (*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*) − (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*)) ≤
nlength σ − *Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*)
using *9 diff-le-mono* **by** (*metis enat-minus-mono1 idiff-enat-enat*)
have *13*: ∃ *i* ≤ *nlength σ* − *Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*).
(*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*) = (*Suc* (*i* + (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*)))
using *11 12* **by** *auto*
have *14*: (∃ *i*≤*nlength σ* − *Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*).
f (*ndropn* (*Suc* (*i* + (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*))) *σ*))
using *13 5* **by** *auto*
have *15*: (*ndropn* (*Suc 0*) (*nmap* (λ*s*. *nnth s 0*) (*nfilter f* (*ndropns σ*)))) =
(*nmap* (λ*s*. *nnth s 0*)
(*nfilter f* (*ndropns* (*ndropn* (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*)) *σ*))))
using *2 8*
by (*metis One-nat-def ileI1 nfilter-ndropns-nmap nkfilter-nlength one-eSuc one-enat-def*)
have *16*: *g* (*nmap* (λ*s* . *nnth s 0*)
(*nfilter f* (*ndropns* (*ndropn* (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*)) *σ*))))
using *1 15* **by** *auto*
have *17*: ∀ *j*< (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*). ¬ *f* (*ndropn j σ*)
using *2 31 nkfilter-not-before*[*of* (*ndropns σ*) *f*]
by (*metis enat-ord-simps*(*1*) *less-imp-le ndropns-nnth order.trans*)
have *18*: (∃ *k*≤*nlength σ*.
f (*ndropn k σ*) ∧
k < *nlength σ* ∧
(∃ *i*≤*nlength σ* − *Suc k*. *f* (*ndropn* (*Suc* (*i* + *k*)) *σ*)) ∧
g (*nmap* (λ*s*. *nnth s 0*) (*nfilter f* (*ndropns* (*ndropn* (*Suc k*) *σ*)))) ∧
(∀ *j*<*k*. ¬ *f* (*ndropn j σ*)))
using *14 16 17 4 7 Suc-ile-eq less-imp-le* **by** *blast*
show *?thesis* **using** *18* **by** (*simp add*: *PiNextsem2*)
qed

lemma *PiNextsem4*:
assumes (*σ* ⊨ (¬ *f*) 𝒰 (*f* ∧ ○(*f* Π *g*)))
shows (*σ* ⊨ *f* Π (○ *g*))
proof −
have *1*: (∃ *k*≤*nlength σ*.
f (*ndropn k σ*) ∧
k < *nlength σ* ∧
(∃ *i*≤*nlength σ* − *Suc k*. *f* (*ndropn* (*Suc* (*i* + *k*)) *σ*)) ∧
g (*nmap* (λ*s*. *nnth s 0*) (*nfilter f* (*ndropns* (*ndropn* (*Suc k*) *σ*)))) ∧
(∀ *j*<*k*. ¬ *f* (*ndropn j σ*)))
using *assms* **by** (*simp add*: *PiNextsem2*)
obtain *k* **where** *2*: *k*≤*nlength σ* ∧ *f* (*ndropn k σ*) ∧ *k* < *nlength σ* ∧ (∃ *i*≤*nlength σ* − *Suc k*.
f (*ndropn* (*Suc* (*i* + *k*)) *σ*)) ∧

```
      g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn (Suc k) σ)))) ∧ (∀ j<k. ¬ f (ndropn j σ))
   using 1 by auto
 have 3: ∃ x ∈ nset (ndropns σ). f x
   using 1 using in-nset-ndropns by auto
 have 4: ∀ x ∈ nset(nkfilter f 0 (ndropns σ)). f (nnth (ndropns σ) x)
   using 3 nkfilter-holds by fastforce
 have 41: (nnth (nkfilter f 0 (ndropns σ)) 0) ≤ nlength σ
   by (metis 3 gen-nlength-def le-cases le-zero-eq ndropns-nlength nkfilter-upperbound nlength-code
     zero-enat-def)
 have 5: f (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) σ)
   by (metis 4 41 i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def)
 have 6: 0 < nlength (nfilter f (ndropns σ))
   using 2 3 nfilter-nlength-zero-conv-a[of (ndropns σ) f]
   by simp
      (metis (no-types, lifting) add.commute eSuc-enat enat.simps(3) enat-add-sub-same
       enat-less-enat-plusI2 ileI1 iless-Suc-eq less-add-Suc2 less-eqE ndropn-Suc-conv-ndropn
       ndropn-nlength ndropns-nlength ndropns-nnth nlength-NCons)
 have 61: (nnth (nkfilter f 0 (ndropns σ)) 1) ∈ nset(nkfilter f 0 (ndropns σ))
   by (metis 3 6 ileI1 in-nset-conv-nnth nkfilter-nlength one-eSuc one-enat-def)
 have 62: (nnth (nkfilter f 0 (ndropns σ)) 1) ≤ nlength (ndropns σ)
   by (metis 3 61 gen-nlength-def in-nset-conv-nnth nkfilter-upperbound nlength-code)
 have 7: f (ndropn (nnth (nkfilter f 0 (ndropns σ)) 1) σ)
   using 4 61 62 by (metis ndropns-nlength ndropns-nnth)
 have 8: (∃ i≤nlength σ. f (ndropn i σ))
   using 1 by blast
 have 9: g (ndropn (Suc 0) (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))
   using 2 3 5 6 nkfilter-not-before[of (ndropns σ) f] nfilter-ndropns-nmap[of σ f 0]
   by (metis One-nat-def ileI1 ndropns-nnth not-less-iff-gr-or-eq one-eSuc one-enat-def)
 have 10: ((∃ i. (enat i) ≤nlength σ ∧ f (ndropn i σ)) ∧
           0 < nlength (nfilter f (ndropns σ)) ∧
           g (ndropn (Suc 0) (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))))
   using 6 8 9 by blast
 show ?thesis using 10
   by (simp add: PiNextsem1)
qed

lemma PiNextsem:
  (σ ⊨ f Π (○ g) = (¬ f) 𝒰 (f ∧ ○(f Π g)))
using PiNextsem3 PiNextsem4 unl-lift2 by blast

lemma PiNext:
⊢ f Π (○ g) = (¬ f) 𝒰 (f ∧ ○(f Π g))
using PiNextsem Valid-def by blast
```

PiUntil

```
lemma PiUntilDistsem1:
  (σ ⊨ f Π (g 𝒰 h)) =
   ((∃ i≤nlength σ. f (ndropn i σ)) ∧
    (∃ k≤nlength (nfilter f (ndropns σ)).
```

$h\ (ndropn\ k\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma))))\ \wedge$
$(\forall j{<}k.\ g\ (ndropn\ j\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))))$
by (*simp add*: *pi-d-def pifilt-def sfxfilt-def until-d-def*)

lemma *PiUntilDistsem2*:
$(\sigma \models (\ f\ \Pi\ g\ )\ \mathcal{U}\ (\ f\ \Pi\ h\ )\ ) =$
$(\exists k{\leq}nlength\ \sigma.$
$(\exists i{\leq}nlength\ \sigma - k.\ f\ (ndropn\ (i + k)\ \sigma)) \wedge$
$h\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall j{<}k.\ (\exists i{\leq}nlength\ \sigma - j.\ f\ (ndropn\ (i + j)\ \sigma)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))))))$
by (*simp add*: *until-d-def pi-d-def pifilt-def sfxfilt-def ndropn-ndropn add.commute*)

lemma *cover*:
assumes $(\exists i{<}k.\ ff(i) {<}(j{::}nat) \wedge j{\leq}\ ff(Suc\ i))$
$(\forall i{<}k.\ ff(i){<}ff(Suc\ i))$
shows $ff(0){<}j \wedge j \leq ff(k)$
using *assms*
proof (*induct k arbitrary*:*j*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
proof −
have *1*: $(\exists i{<}k.\ ff(i) {<}(j{::}nat) \wedge j{\leq}\ ff(Suc\ i)) \vee (ff\ (k) {<}j \wedge j{\leq}ff(Suc\ k))$
using *Suc.prems*(*1*) *less-SucE* **by** *blast*
have *2*: $(\forall i{<}k.\ ff(i){<}ff(Suc\ i))$
using *Suc.prems*(*2*) **by** *auto*
have *3*: $ff\ k < ff\ (Suc\ k)$
by (*simp add*: *Suc.prems*(*2*))
have *4*: $(ff(0){<}j \wedge j \leq ff(k)) \vee\ (ff\ (k) {<}j \wedge j{\leq}ff(Suc\ k))$
using *1 2 Suc.hyps* **by** *blast*
have *41*: $ff(0){<}j$
using *2 4 Suc.hyps order-refl* **by** *force*
have *42*: $j \leq ff(Suc\ k)$
using *3 4* **by** *linarith*
have *5*: $ff(0){<}j \wedge j \leq ff(Suc\ k)$
by (*simp add*: *41 42*)
show *?thesis*
by (*simp add*: *5*)
qed
qed

lemma *cover-a*:
assumes $(\forall\ j.$
$(j{\leq}\ ff\ 0) \vee (\exists i{<}k.\ ff(i) {<}(j{::}nat) \wedge j{\leq}\ ff(Suc\ i))$
$\longrightarrow gg\ j)$

$(\forall i{<}k.\ ff(i){<}ff(Suc\ i))$

```
 shows   (∀ j< ff k. gg j)
proof −
 have 1: (∀ j< ff 0. gg j)
   by (simp add: assms(1))
 have 2: (∀ j. ff 0 < j ∧ j≤ ff k ⟶ gg j)
   proof
     fix j
     show ff 0 < j ∧ j ≤ ff k ⟶ gg j
      using assms
       proof (induct k arbitrary: j)
       case 0
       then show ?case by simp
       next
       case (Suc k)
       then show ?case
         proof −
          have 21: (∀ j. (j≤ ff 0) ∨ (∃ i<k. ff(i) <(j::nat) ∧ j≤ ff(Suc i))
                    ∨ (ff k < j ∧ j ≤ ff (Suc k)    ) ⟶ gg j)
             using Suc.prems(1) less-SucI by blast
          have 22: (∀ i<k. ff(i)<ff(Suc i))
            using Suc.prems(2) by auto
          have 23: ff k < ff (Suc k)
            by (simp add: Suc.prems(2))
          have 24: (∀ j.
                    (j≤ ff 0) ∨ (ff 0 <j ∧ j ≤ ff k)
                     ∨ (ff k < j ∧ j ≤ ff (Suc k)    )
                   ⟶ gg j)
            using 21 22 Suc.hyps by blast
          have 25: ff 0 < j ∧ j ≤ ff (Suc k) ⟶ gg j
            using 24 not-less by blast
          show ?thesis using 25 by blast
         qed
       qed
   qed
  show ?thesis
  by (metis 2 assms(1) less-or-eq-imp-le linorder-neqE-nat)
qed

lemma PiUntilDistsem3:
 assumes (σ ⊨ f Π (g 𝒰 h))
 shows   (σ ⊨ ( f Π g ) 𝒰 ( f Π h ) )
proof −
  have 1: ((∃ i≤nlength σ. f (ndropn i σ)) ∧
           (∃ k≤nlength (nfilter f (ndropns σ)).
             h (ndropn k (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))) ∧
             (∀ j<k. g (ndropn j (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))))))
    using assms PiUntilDistsem1 by blast
  have 2: ∃ x ∈ nset(ndropns σ). f x
    using 1 by (simp add: sfxfilter-help)
  have 3: ∀ x ∈ nset(nkfilter f 0 (ndropns σ)). f (nnth (ndropns σ) x)
```

using *2 nkfilter-holds* **by** *fastforce*
have *4*: $(\exists k{\leq}nlength\ (nfilter\ f\ (ndropns\ \sigma)).$
$h\ (ndropn\ k\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))) \wedge$
$(\forall j{<}k.\ g\ (ndropn\ j\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma))))))$
using *1* **by** *auto*
obtain *k* **where** *5*: $k{\leq}nlength\ (nfilter\ f\ (ndropns\ \sigma)) \wedge$
$h\ (ndropn\ k\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))) \wedge$
$(\forall j{<}k.\ g\ (ndropn\ j\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
using *4* **by** *auto*
have *51*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k) \leq nlength\ \sigma$
by (*metis* (*mono-tags*, *lifting*) *2 5 gen-nlength-def ndropns-nlength nkfilter-nlength nkfilter-upperbound nlength-code*)
have *6*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k)\ \ \sigma)$
using *2 3 5*
by (*metis 1 ndropns-nfilter-nnth nlength-nmap pifilt-def sfxfilt-def sfxfilt-pifilt-nnth-ndropn*)
have *7*: $k{=}0 \longrightarrow$
$(\exists i.\ (enat\ i){\leq}nlength\ \sigma - k \wedge f\ (ndropn\ (i + k)\ \sigma)) \wedge$
$h\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ \ (ndropns\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall j{<}k.\ (\exists i{\leq}nlength\ \sigma - j.\ f\ (ndropn\ (i + j)\ \sigma)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma)))))$
using *1 5* **by** *auto*
have *71*: $k{>}0 \longrightarrow (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k - 1)) \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma))$
by (*metis 2 5 One-nat-def Suc-ile-eq Suc-pred in-nset-conv-nnth less-imp-le nkfilter-nlength*)
have *72*: $k{>}0 \longrightarrow enat\ (k - 1) \leq nlength\ (nkfilter\ f\ 0\ (ndropns\ \sigma))$
by (*metis 2 5 One-nat-def Suc-ile-eq Suc-pred nkfilter-nlength order-less-imp-le*)
have *8*: $k{>}0 \longrightarrow f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k - 1))\ \ \sigma)$
using *3 2 71 72 nkfilter-upperbound*[*of* $(ndropns\ \sigma)\ f\ k{-}1\ 0$]
by (*metis add-0 ndropns-nlength ndropns-nnth zero-enat-def*)
have *9*: $k{>}0 \longrightarrow (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k - 1)) < (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k)$
by (*metis 2 5 One-nat-def Suc-pred nkfilter-mono nkfilter-nlength*)
have *10*: $k{>}0 \longrightarrow\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k - 1)) \leq nlength\ \sigma$
by (*metis 2 71 gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code*)
have *11*: $k{>}0 \longrightarrow (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k) \leq nlength\ \sigma$
using *nkfilter-upperbound*[*of ndropns* $\sigma\ f\ k\ 0$]
using *51* **by** *blast*
have *12*: $k{>}0 \longrightarrow$
$h\ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k)\ \sigma))))$
using *nfilter-nkfilter-ndropn*[*of f* $(ndropns\ \sigma)\ k$]
ndropns-nfilter-ndropn-a[*of k f* σ]
by (*metis 2 5 ndropn-nmap*)
have *121*: $k{>}0 \longrightarrow$
$h\ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k{-}1)))\ \sigma))))$
by (*metis 2 5 One-nat-def Suc-pred nfilter-ndropns-nmap*)
have *13*: $k{>}0 \longrightarrow (\exists i{\leq}nlength\ \sigma - (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k).$
$f\ (ndropn\ (i + (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k))\ \sigma))$
using *6* **by** (*metis add.left-neutral zero-enat-def zero-le*)
have *130*: $k{>}0 \longrightarrow (\exists\ i.\ i{+}\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k{-}1))) \leq nlength\ \sigma\ \ \wedge$

$(ndropn\ (i+(Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))))\ \sigma) =$
$(ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k)\ \sigma))$
using *9* **by** (*metis 11 Suc-leI diff-add*)
have *131*: $k>0 \longrightarrow (\exists\, i.\ i+\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1)))\leq nlength\ \sigma\ \wedge$
$f\ (ndropn\ (i\ +\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))))\ \sigma))$
using *10 6 9 11 130* **by** *auto*
have *132*: $k>0 \longrightarrow (\exists\, i\leq nlength\ \sigma\ -\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))).$
$f\ (ndropn\ (i\ +\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))))\ \sigma))$
using *131*
by (*metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat*)
have *14*: $k>0 \longrightarrow (\forall\ j< (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k).$
$(\exists\, i.\ i+j\leq\ nlength\ \sigma \wedge f\ (ndropn\ (i\ +\ j)\ \sigma)))$
using *131* **by** (*metis 11 6 diff-add less-imp-le plus-enat-simps(1)*)
have *141*: $k>0 \longrightarrow (\forall\ j< (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))).$
$(\exists\, i.\ i+j\leq nlength\ \sigma \wedge f\ (ndropn\ (i\ +\ j)\ \sigma)))$
using *14 9* **by** *auto*
have *142*: $k>0 \longrightarrow (\forall\ j< (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))).$
$(\exists\, i\leq nlength\ \sigma\ -j.\quad f\ (ndropn\ (i\ +\ j)\ \sigma)))$
using *141* **by** (*metis add.commute enat.simps(3) enat-add-sub-same enat-minus-mono1*)
have *15*: $(\forall\, j<k.\ g\ (ndropn\ j\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
using *5* **by** *blast*
have *151*: $(\forall\, j<k.\ g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (ndropn\ j\ \ (nfilter\ f\ (ndropns\ \sigma)))))$
by (*metis 15 ndropn-nmap*)
have *152*: $(\forall\, j<k.\ (ndropn\ j\ \ (nfilter\ f\ (ndropns\ \sigma))) =$
$(nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ (ndropns\ \sigma))))$
using *nfilter-nkfilter-ndropn-1*[*of ndropns* σ *f*]
by (*simp add*: *2 5 less-imp-le nkfilter-nlength order-less-le-subst2*)
have *16*: $(\forall\, j<k.\ g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ (ndropns\ \sigma)))))$
using *151 152 nfilter-nkfilter-ndropn* **by** *simp*
have *1610*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k) \leq nlength(ndropns\ \sigma)$
by (*simp add*: *51 ndropns-nlength*)
have *1611*: $(\forall\, j<k.\ \ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j) <\ \ \ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k))$
by (*simp add*: *2 5 nidx-nkfilter-gr nkfilter-nlength*)
have *1612*: $(\forall\, j<k.\ \ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j) \leq nlength(ndropns\ \sigma))$
using *1610 1611* **by** (*meson enat-ord-simps(2) less-imp-le order-less-le-subst2*)
have *161*: $(\forall\, j<k.\ (\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ (ndropns\ \sigma))) =$
$(\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ \sigma))))$
using *1612* **by** (*simp add*: *ndropn-ndropns*)
have *17*: $(\forall\, j<k.\ g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ \sigma)))))$
using *16 161* **by** *auto*
have *18*: $k>0 \longrightarrow$
$g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))\ \sigma))))$
using *17* **by** *simp*
have *19*: $k>0 \longrightarrow$
$g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma))))$
using *17* **by** *blast*

have *20*: $k{>}0 \longrightarrow (nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma))) =$
$(nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ (ndropns\ \sigma)))$
using *161* **by** *auto*
have *200*: $nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0 \leq nlength\ (ndropns\ \sigma)$
using *1610 1612* **by** *blast*
have *21*: $k{>}0 \longrightarrow (\forall\ j{\leq}\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0).$
$(\ (ndropns\ (ndropn\ j\ \sigma))) =$
$(\ (ndropn\ j\ (ndropns\ \sigma))))$
using *200 ndropn-ndropns* **by** (*simp add*: *ndropn-ndropns order-subst2*)
have *22*: $k{>}0 \longrightarrow (\forall\ j{\leq}\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma)\ )\ 0).$
$(nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))) =$
$(nfilter\ f\ (ndropn\ j\ (ndropns\ \sigma))))$
using *21* **by** *auto*
have *23*: $k{>}0 \longrightarrow$
$(\forall\ j{\leq}\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma)\ )\ 0).$
$(nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma)))) =$
$(nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))))$
$)$
by (*simp add*: *2 22 nfilter-ndropns-nmap-help-0*)
have *24*: $k{>}0 \longrightarrow$
$(\forall\ j{\leq}\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma)\ )\ 0).$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma)))) =$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))))$
$)$
using *23* **by** *auto*
have *241*: $k{>}0 \longrightarrow (\forall\ j{\leq}\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma)\ \ )\ 0).$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))))\ )$
using *19 24* **by** *blast*
have *25*: $k{>}0 \longrightarrow (\forall\ i < k - 1.$
$(\forall\ l.\ l{\leq}\ \ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (Suc\ i)\ ) \wedge$
$(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ i\ ) < l \longrightarrow$
$(nfilter\ f\ (ndropn\ (nnth(nkfilter\ f\ 0\ (ndropns\ \sigma))\ (Suc\ i))\ (ndropns\ \sigma)) =$
$(nfilter\ f\ (ndropn\ l\ (ndropns\ \ \sigma)))\quad\ \ )\ )\ \ )$
proof *auto*
fix *i*
fix *l*
assume *a0*: $0 < k$
assume *a1*: $i < k - Suc\ 0$
assume *a2*: $l \leq nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (Suc\ i)$
assume *a3*: $nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ i < l$
show $nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (Suc\ i))\ (ndropns\ \sigma)) =$
$nfilter\ f\ (ndropn\ l\ (ndropns\ \sigma))$
proof −
have *251*: $k{=}1 \Longrightarrow$
$nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (Suc\ i))\ (ndropns\ \sigma)) =$
$nfilter\ f\ (ndropn\ l\ (ndropns\ \sigma))$
using *a1* **by** *force*
have *2511*: $1 < k \Longrightarrow enat\ (Suc\ i) \leq nlength\ (nfilter\ f\ (ndropns\ \sigma))$

```
      by (metis (mono-tags, opaque-lifting) 5 One-nat-def Suc-diff-1 Suc-mono a0 a1
          enat-ord-simps(1) less-imp-le order-subst2)
    then have 252: 1<k ⟹
          nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
          nfilter f (ndropn l (ndropns σ))
     using 2 5 a1 nfilter-ndropns-nmap-help-j[of ndropns σ f l i] a2 a3 by fastforce
    show ?thesis
    using 252 a1 by fastforce
   qed
  qed
have 261: k>0 ⟶ (∀ i < k −1.
              (∀ l. l≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i) ) ∧
                   (nnth (nkfilter f 0 (ndropns σ)) i ) < l ⟶
                   l ≤ nlength (ndropns σ) ))
  using 1612 by (meson diff-less enat-ord-simps(1) less-one less-trans-Suc order-subst2)
have 262: k>0 ⟶ (∀ i < k −1.
              (∀ l. l≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i) ) ∧
                   (nnth (nkfilter f 0 (ndropns σ)) i ) < l ⟶
                   (ndropn l (ndropns σ)) = (ndropns (ndropn l σ)) ))
  using 261 by (simp add: ndropn-ndropns)
have 26: k>0 ⟶ (∀ i < k −1.
              (∀ l. l≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i) ) ∧
                   (nnth (nkfilter f 0 (ndropns σ)) i ) < l ⟶
               (nmap (λs. nnth s 0)
                 (nfilter f (ndropn (nnth(nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ))))=
               (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn l σ))) ) ) )
  using 25 262 by auto
have 27: k>0 ⟶ (∀ i < k −1.
              (∀ l. l≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i) ) ∧
                   (nnth (nkfilter f 0 (ndropns σ)) i ) < l ⟶
              g (nmap (λs. nnth s 0)
                 (nfilter f
                    (ndropn (nnth(nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ))))))
  by (simp add: 16)
have 28: k>0 ⟶ (∀ i < k −1.
              (∀ l. l≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i) ) ∧
                   (nnth (nkfilter f 0 (ndropns σ)) i ) < l ⟶
              g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn l σ))) )))
  using 25 262 27 by auto
have 281: k>0 ⟶
         (∀ j.
            (j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0 ) ∨
            (∃ i. i < k −1 ∧
                j≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i) ) ∧
                (nnth (nkfilter f 0 (ndropns σ)) i ) < j) )
                ⟶ g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ))) ) ))
  using 241 28 by blast
have 282: k>0 ⟶ (∀ i<k − 1.
                  nnth (nkfilter f 0 (ndropns σ)) i < nnth (nkfilter f 0 (ndropns σ)) (Suc i))
  using nidx-nkfilter-expand[of ndropns σ f - 0]
```

```
  by (metis (full-types) 2 72 Suc-ile-eq enat-ord-simps(2) order-less-le-subst2)
 have 29: k>0 ⟶ (∀ j< (Suc (nnth (nkfilter f 0 (ndropns σ)) (k−1))).
                  g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  using 281 282 cover-a[of λ x. nnth (nkfilter f 0 (ndropns σ)) x  k−1
                        (λ j. g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))]
  using 18 less-antisym by blast
 have 30: k>0 ⟶ (∃ k≤nlength σ.
    (∃ i≤nlength σ − k. f (ndropn (i + k) σ)) ∧
    h (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn k σ)))) ∧
    (∀ j<k. (∃ i≤nlength σ − j. f (ndropn (i + j) σ)) ∧
          g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ))))))
  using 10 11 121 132 142 29 9 by simp
      (metis add-Suc-right antisym-conv2 eSuc-enat enat-ord-simps(1) ileI1 leD)
 have 31: (∃ k≤nlength σ.
           (∃ i≤nlength σ − k. f (ndropn (i + k) σ)) ∧
             h (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn k σ)))) ∧
             (∀ j<k. (∃ i≤nlength σ − j. f (ndropn (i + j) σ)) ∧
                g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ))))))
  using 30 7 by (metis not-gr-zero zero-enat-def zero-le)
 show ?thesis using 31 by (simp add: PiUntilDistsem2)
qed

lemma PiUntilDistsem4:
 assumes (σ ⊨ ( f Π g ) 𝒰 ( f Π h ) )
 shows  (σ ⊨ f Π (g 𝒰 h))
proof −
 have 1: (∃ k≤nlength σ.
          (∃ i≤nlength σ − k. f (ndropn (i + k) σ)) ∧
          h (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn k σ)))) ∧
          (∀ j<k.
            (∃ i≤nlength σ − j. f (ndropn (i + j) σ)) ∧
            g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ))))))
  using assms by (simp add: PiUntilDistsem2)
 obtain k where 2: k≤nlength σ ∧
                   (∃ i≤nlength σ − k. f (ndropn (i + k) σ)) ∧
                   h (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn k σ)))) ∧
                   (∀ j<k.
                     (∃ i≤nlength σ − j. f (ndropn (i + j) σ)) ∧
                     g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  using 1 by auto
 have 3: (∃ i≤nlength σ − k. f (ndropn (i + k) σ))
  using 2 by auto
 have 31: (∃ i≤ nlength (ndropns (ndropn k σ)). f (nnth (ndropns (ndropn k σ)) i))
  by (metis 3 add.commute ndropn-ndropn ndropn-nlength ndropns-nlength ndropns-nnth)
 have 4: k ≤ nlength σ
  using 2 by auto
 obtain i where 5: (enat i)≤nlength (ndropns (ndropn k σ))∧ f (nnth (ndropns (ndropn k σ)) i) ∧
                   i = (LEAST na. enat na ≤ nlength (ndropns (ndropn k σ)) ∧
                   f (nnth (ndropns (ndropn k σ)) na))
  by (metis (no-types, lifting) 31 LeastI-ex)
```

have *51*: $i = (LEAST\ na.\ enat\ na \leq nlength\ (ndropns\ (ndropn\ k\ \sigma)) \wedge f\ (nnth\ (ndropns\ (ndropn\ k\ \sigma))\ na))$
using *5* **by** *auto*
have *60*: $(enat\ i) \leq nlength\ \sigma - k$
by (*metis 5 ndropn-nlength ndropns-nlength*)
have *601*: $k+i \leq nlength\ \sigma$
by (*metis 4 60 dual-order.eq-iff enat.simps(3) enat-add-sub-same enat-less-enat-plusI2 less-eqE less-imp-le order.not-eq-order-implies-strict plus-enat-simps(1)*)
have *6*: $i+k \leq nlength\ \sigma$
by (*metis 601 add.commute*)
have *61*: $\exists x \in nset\ (ndropns\ (ndropn\ k\ \sigma)).\ f\ x$
using *31 exists-Pred-nnth-nset* **by** *blast*
have *62*: $\exists x \in nset\ (ndropns\ (ndropn\ (k+i)\ \sigma)).\ f\ x$
by (*metis 5 in-nset-conv-nnth ndropn-ndropn ndropn-ndropns nnth-zero-ndropn zero-enat-def zero-le*)
have *7*: $(\exists i \leq nlength\ \sigma.\ f\ (ndropn\ i\ \sigma))$
by (*metis 5 6 add.commute ndropn-ndropn ndropns-nlength ndropns-nnth*)
have *71*: $\exists x \in nset\ (ndropns\ \sigma).\ f\ x$
using *5 6* **using** *in-nset-ndropns* **using** *7* **by** *blast*
have *72*: $\exists\ x \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma)).\ f\ (nnth\ (ndropns\ \sigma)\ x)$
using *71 nkfilter-holds-b[of (ndropns σ) f] sfxfilter-help[of σ]*
by (*metis 71 Nat.add-0-right ndropns-nlength ndropns-nnth*)
have *722*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0) \leq nlength\ (ndropns\ (ndropn\ k\ \sigma))$
by (*metis 61 gen-nlength-def nkfilter-upperbound nlength-code zero-enat-def zero-le*)
have *723*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0) \leq nlength\ (\ (ndropn\ k\ \sigma))$
by (*metis 722 ndropns-nlength*)
have *73*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0)\ \ (ndropn\ k\ \sigma))$
using *nkfilter-holds[of ndropns (ndropn k σ) f 0]*
by (*metis (no-types, lifting) 61 722 diff-zero ndropns-nfilter-nnth ndropns-nlength ndropns-nnth nkfilter-nfilter zero-enat-def zero-le*)
have *74*: $f\ (ndropn\ ((nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0)+k)\ \ \sigma)$
using *73* **by** (*simp add*: *add.commute ndropn-ndropn*)
have *75*: $f\ (ndropn\ k\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0)\ \sigma))$
by (*simp add*: *74 ndropn-ndropn*)
have *76*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0) = nleast\ f\ (ndropns\ (ndropn\ k\ \sigma))$
by (*simp add*: *61 nkfilter-nleast*)
have *77*: $nleast\ f\ (ndropns\ (ndropn\ k\ \sigma)) = (LEAST\ na.\ enat\ na \leq nlength\ (ndropns\ (ndropn\ k\ \sigma)) \wedge f\ (nnth\ (ndropns\ (ndropn\ k\ \sigma))\ na))$
using *61 nleast-conv[of ndropns (ndropn k σ) f]* **by** *auto*
have *78*: $nleast\ f\ (ndropns\ (ndropn\ k\ \sigma)) = i$
using *5 77* **by** *blast*
have *8*: $h\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ k\ \sigma))))$
using *2* **by** *auto*
have *10*: $nfirst\ (nkfilter\ f\ k\ (ndropn\ k\ (ndropns\ \sigma))) = ((nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0)+k)$
by (*metis 2 61 diff-add ndropn-0 ndropn-ndropns ndropn-nfirst ndropns-nlength nkfilter-lowerbound nkfilter-nnth-n-zero zero-enat-def zero-le*)
have *101*: $nfirst\ (nkfilter\ f\ k\ (ndropn\ k\ (ndropns\ \sigma))) = k+ i$
by (*simp add*: *10 76 78*)
have *90*: $f\ (nlast\ (ntaken\ ((nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0)+k)\ (ndropns\ \sigma)))$
by (*metis 6 74 76 78 ndropns-nnth ntaken-nlast*)

have *91*: ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) ≤ *nlength* (*ndropns σ*)
using *nkfilter-upperbound*[*of* (*ndropns* (*ndropn k σ*)) *f 0 0*]
ndropn-nlength[*of k σ*] *ndropns-nlength*[*of ndropn k σ*]
by (*simp add*: *6 76 78 ndropns-nlength*)
have *92*: ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) = ((*nnth* (*nkfilter f k* (*ndropns* (*ndropn k σ*))) *0*))
by (*simp add*: *61 nkfilter-nleast*)
let *?kf* = (*the-enat* ((*nlength*(*nfilter f* (*ntaken* ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) (*ndropns σ*))))))
let *?nkf* = (*the-enat* ((*nlength*(*nkfilter f 0* (*ntaken* ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) (*ndropns σ*))))))
let *?pkf* = (*the-enat* ((*nlength* (*nkfilter f 0* (*ntaken k* (*ndropns σ*))))))
have *93*: *?kf* = *?nkf*
by (*metis 90 nfinite-ntaken nkfilter-nlength nset-nlast*)
have *94*: ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) ≤ *nnth* (*nkfilter f 0* (*ndropns σ*)) *?nkf*
by (*metis 74 90 91 add.left-neutral eq-iff ndropn-nfirst ndropns-nlength ndropns-nnth*
nkfilter-chop1-ndropn nkfilter-nfirst)
have *95*: (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) = *0* ⟹ *k* = *nnth* (*nkfilter f 0* (*ndropns σ*)) *?nkf*
by (*metis 10 90 91 add-cancel-left-left ndropn-nfirst nkfilter-chop1-ndropn*)
have *98*: *0*< *?nkf* ∧ *0*< (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) ⟹
nnth (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*) < *k*
proof −
assume *b*: *0*< *?nkf* ∧ *0*< (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)
have *980*: (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*)) ∈ *nset* (*nkfilter f 0* (*ndropns σ*))
by (*metis in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *981*: *f* (*nnth* (*ndropns σ*) (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*)))
using *nkfilter-holds*[*of* (*ndropns σ*) *f* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*)) *0*]
using *71 980* **by** *auto*
have *982*: ¬ *f* (*ndropn k σ*)
by (*metis 10 4 add-cancel-left-left b gr-implies-not0 ndropn-nfirst ndropns-nnth*
nkfilter-nfirst)
have *984*: *f* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*)) *σ*)
using *71 980 981 ndropns-nnth*[*of* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*)) *σ*]
by (*metis gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code*)
have *985*: ⋀*j*. *k* ≤*j* ⟹ *j*< *k*+*i* ⟹ ¬ *f* (*ndropn j σ*)
proof −
fix *j*
assume *a0*: *k*≤*j*
assume *a1*: *j*<*k*+*i*
show ¬ *f* (*ndropn j σ*)
proof −
have *9851*: ∃ *x*∈*nset* (*ndropn k* (*ndropns σ*)). *f x*
by (*simp add*: *4 61 ndropn-ndropns ndropns-nlength*)
have *9852*: *enat k* ≤ *nlength* (*ndropns σ*)
by (*simp add*: *4 ndropns-nlength*)
have *9853*: *k* ≤ *j*
by (*simp add*: *a0*)
have *9854*: *j* < *nnth* (*nkfilter f k* (*ndropn k* (*ndropns σ*))) *0*
using *10 101 92 9852 a1 ndropn-ndropns* **by** *fastforce*
have *9855*: ¬ *f* (*nnth* (*ndropns σ*) *j*)

```
        using 9851 9852 9853 9854 nkfilter-n-not-before[of k ndropns σ f j] by auto
      show ?thesis
        by (metis 10 101 6 76 78 9855 a1 enat-ord-simps(2) less-imp-le
            ndropns-nnth order-less-le-subst2)
    qed
  qed
  have 986: ntaken ?nkf (nkfilter f 0 (ndropns σ)) =
          nkfilter f 0 (ntaken (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k) (ndropns σ))
    using nkfilter-chop1-ntaken[of (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k) (ndropns σ) f 0]
    using 90 91 by blast
  have 987: ¬ enat (?nkf) ≤ nlength (nkfilter f 0 (ndropns σ)) ∨
        nnth (nkfilter f 0 (ndropns σ)) (?nkf − 1) < nnth (nkfilter f 0 (ndropns σ)) (?nkf)
    using 71 by (meson b diff-less less-one nidx-nkfilter-gr)
  have 988: nlast (nkfilter f 0 (ntaken (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k) (ndropns σ)))
=
          0 + (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k)
    by (simp add: 90 91 nkfilter-nlast)
  have 989: nnth (nkfilter f 0 (ndropns σ)) (?nkf) = nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k
    using 986
    by (metis 988 add.left-neutral ntaken-nlast)
  have 990: nnth (nkfilter f 0 (ndropns σ)) (?nkf) = nfirst (nkfilter f k (ndropn k (ndropns σ)))
    using 10 989 by fastforce
  have 991: nnth (nkfilter f 0 (ndropns σ)) (?nkf) = k + i
    using 10 101 989 by presburger
  show ?thesis
    by (metis 984 985 986 987 991 enat-the-enat infinity-ileE le-cases not-le-imp-less ntaken-all)
qed
have 100: h (nmap (λs. nnth s 0) (ndropn ?kf (nfilter f (ndropns σ))))
  using ndropns-nfilter-ndropn-a[of - f ndropn k σ ]
    nfilter-chop1-ndropn[of ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ) f ]
    101 61 76 78 8 90 91
  by simp
    (metis 10 ndropn-0 ndropn-ndropn ndropn-ndropns zero-enat-def zero-le)
have 11: h (ndropn ?kf (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))
  by (simp add: 100 ndropn-nmap)
have 111: ?kf ≤ nlength (nfilter f (ndropns σ))
  by (metis 90 enat-ile le-cases nfilter-chop1-ntaken ntaken-all the-enat.simps)
have 112: nfinite (nfilter f (ntaken ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ)))
  by (metis 90 91 nfilter-chop1-ntaken nfinite-ntaken)
have 13: (∀ j<k.
        (∃ x ∈ nset(ndropns (ndropn j σ)). f x) ∧
        g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  by (metis 2 add.commute in-nset-ndropns ndropn-ndropn ndropn-nlength)
have 151: (∀ jj<?kf.
        (ndropn jj (nfilter f (ndropns σ))) =
        nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) jj) σ))
      ) using nfilter-nkfilter-ndropn
  by (simp add: 111 71 less-imp-le ndropns-nfilter-ndropn-a order-less-le-subst2)
have 152: (⋀ jj. (enat jj)<?kf ⟹ (nnth (nkfilter f 0 (ndropns σ)) jj) < k )
  proof −
```

fix *jj*
assume *a*: (*enat jj*) < *?kf*
show *nnth* (*nkfilter f 0* (*ndropns σ*)) *jj* < *k*
proof −
have *1522*: (*enat jj*) ≤ *?nkf* −*1*
using *93 a* **by** *auto*
have *1523*: *nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*) < *nnth* (*nkfilter f 0* (*ndropns σ*)) *?nkf*
by (*metis 111 71 93 a diff-less gr-implies-not-zero less-numeral-extra*(*1*)
nidx-nkfilter-gr nkfilter-nlength not-gr-zero zero-enat-def)
have *15230*: *enat* (*?nkf* − *1*) ≤ *nlength* (*nkfilter f 0* (*ndropns σ*))
by (*metis 111 71 93 One-nat-def Suc-ile-eq Suc-pred a less-imp-le nkfilter-nlength*
not-gr-zero not-less-zero zero-enat-def)
have *1524*: *nnth* (*nkfilter f 0* (*ndropns σ*)) *jj* ≤ *nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf*−*1*)
using *1522 1523 71 93 15230*
nidx-less-eq[*of* (*nkfilter f 0* (*ndropns σ*)) (*jj*) *?nkf* −*1*]
nidx-nkfilter[*of* (*ndropns σ*) *f 0*]
using *enat-ord-simps*(*1*) **by** *blast*
have *1525*: *k* ≤ *nnth* (*nkfilter f 0* (*ndropns σ*)) *?nkf*
using *94 add-leE* **by** *blast*
have *1526*: (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) = *0* ⟹
nnth (*nkfilter f 0* (*ndropns σ*)) *jj* < *k*
using *1523 1524 95* **by** *linarith*
have *1527*: *0*<(*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) ⟹
nnth (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*) < *k*
using *93 98 a* **by** *auto*
show *?thesis* **using** *1524 1526 1527 dual-order.strict-trans2* **by** *blast*
qed
qed
have *153*: (∀ *jj*<*?kf*.
g (*nmap* (λ*s*. *nnth s 0*)
(*nfilter f* (*ndropns* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *jj*) *σ*)))
))
using *112 13 152 nfinite-nlength-enat* **by** *force*
have *1530*: (∀ *jj*<*?kf*.
g (*nmap* (λ*s*. *nnth s 0*) (*ndropn jj* (*nfilter f* (*ndropns σ*)))))
by (*simp add*: *151 153*)
have *1531*: (∀ *jj*<*?kf*.
g (*ndropn jj* (*nmap* (λ*s*. *nnth s 0*) (*nfilter f* (*ndropns σ*)))))
by (*simp add*: *1530 ndropn-nmap*)
have *154*: ((∃ *i*. (*enat i*)≤*nlength σ* ∧ *f* (*ndropn i σ*)) ∧
(∃ *kk*. (*enat kk*)≤*nlength* (*nfilter f* (*ndropns σ*)) ∧
h (*ndropn kk* (*nmap* (λ*s*. *nnth s 0*) (*nfilter f* (*ndropns σ*)))) ∧
(∀ *jj*<*kk*. *g* (*ndropn jj* (*nmap* (λ*s*. *nnth s 0*) (*nfilter f* (*ndropns σ*)))))))
using *11 111 1531 7* **by** *blast*
show *?thesis* **by** (*simp add*: *154 PiUntilDistsem1*)
qed

lemma *PiUntilDistsem*:
σ ⊨ *f* Π (*g* 𝒰 *h*) = (*f* Π *g*) 𝒰 (*f* Π *h*)
using *PiUntilDistsem3 PiUntilDistsem4* **using** *unl-lift2* **by** *blast*

lemma *PiUntilDist*:
$\vdash\ f\ \Pi\ (g\ \mathcal{U}\ h) = (\ f\ \Pi\ g\ )\ \mathcal{U}\ (\ f\ \Pi\ h\ )$
using *PiUntilDistsem Valid-def* **by** *blast*

PiChopstar

lemma *wnextboxnotstatesem*:
assumes $k \leq nlength\ \sigma$
shows $(\forall\ j \leq nlength\ \sigma.\ k<j \longrightarrow \neg\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ j)) =$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma)$

using *assms*
proof (*auto simp add*: *itl-defs ndropn-nfirst*)
show $\bigwedge n.\ enat\ k \leq nlength\ \sigma \Longrightarrow$
$\forall j.\ enat\ j \leq nlength\ \sigma \longrightarrow k < j \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ j)) \Longrightarrow$
$nlength\ \sigma - enat\ k \neq enat\ 0 \Longrightarrow$
$enat\ n \leq nlength\ \sigma - enat\ k - enat\ (Suc\ 0) \Longrightarrow w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n)))) \Longrightarrow False$
by (*metis add-Suc-right assms diff-self-eq-0 dual-order.order-iff-strict*
enat-ord-simps(2) idiff-enat-enat less-add-same-cancel1 linorder-le-cases
nfinite-nlength-enat nfinite-ntaken not-le-imp-less ntaken-all ntaken-nlast
zero-enat-def zero-less-Suc)
show $\bigwedge j.\ enat\ k \leq nlength\ \sigma \Longrightarrow enat\ j \leq nlength\ \sigma \Longrightarrow k < j \Longrightarrow w\ (NNil\ (nnth\ \sigma\ j)) \Longrightarrow$
$nlength\ \sigma - enat\ k = enat\ 0 \Longrightarrow False$
by (*metis add.right-neutral enat.inject enat-add-sub-same enat-ord-code(4) leD less-eqE*
min.strict-order-iff min-enat-simps(1) zero-enat-def)
show $\bigwedge j.\ enat\ k \leq nlength\ \sigma \Longrightarrow$
$enat\ j \leq nlength\ \sigma \Longrightarrow$
$k < j \Longrightarrow w\ (NNil\ (nnth\ \sigma\ j)) \Longrightarrow$
$\forall n.\ enat\ n \leq nlength\ \sigma - enat\ k - enat\ (Suc\ 0) \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n)))) \Longrightarrow False$
proof −
fix *j*
assume *a0*: $enat\ k \leq nlength\ \sigma$
assume *a1*: $enat\ j \leq nlength\ \sigma$
assume *a2*: $k < j$
assume *a3*: $w\ (NNil\ (nnth\ \sigma\ j))$
assume *a4*: $\forall n.\ enat\ n \leq nlength\ \sigma - enat\ k - enat\ (Suc\ 0) \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n))))$
show *False*
proof −
have *1*: $j = (Suc\ (k + (j - (Suc\ k))))$
using *a2* **by** *auto*
have *2*: $\forall n.\ enat\ n + 1 + k \leq nlength\ \sigma \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n))))$
by *auto*
(*metis One-nat-def a4 add.commute enat.simps(3) enat-add-sub-same enat-minus-mono1*
one-enat-def)
have *3*: $(j - (Suc\ k)) + 1 + k \leq nlength\ \sigma$
using *1 a1* **by** *simp*
have *4*: $\neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + (j - (Suc\ k))))))$
using *2 3 eSuc-enat plus-1-eSuc(2)* **by** *auto*
from *1 4 a3* **show** *?thesis* **by** *auto*

```
    qed
  qed
qed

lemma NotStateUntilStateAndsem:
 (σ ⊨ (init (¬ w)) 𝒰 ((init w) ∧ f) ) =
   (∃ k. (enat k)≤nlength σ ∧ w (NNil (nnth σ k)) ∧ f (ndropn k σ) ∧ (∀ j<k. ¬ w (NNil (nnth σ j))))

by (auto simp add: until-d-def init-defs ndropn-nfirst)

lemma StateUntilEqvWPrevChopsem:
  σ ⊨ (init w) 𝒰 f = (wprev (□ (init w)))⌢f
proof (auto simp add: until-d-def itl-defs ntaken-nnth ndropn-nfirst  min-absorb1 )
 show ⋀k. enat k ≤ nlength σ ⟹
        f (ndropn k σ) ⟹
        ∀ j<k. w (NNil (nnth σ j)) ⟹
        ∃ n. enat n ≤ nlength σ ∧
            (min (enat n) (nlength σ) = enat 0 ∨
               (∀ na. na ≤ the-enat (min (enat (n − Suc 0)) (epred (nlength σ))) ∧ na ≤ n ∧ enat na ≤
nlength σ ⟶
                    w (NNil (nnth σ na)))) ∧
            f (ndropn n σ)
  by (metis One-nat-def Suc-pred epred-enat epred-min le-imp-less-Suc min.orderE not-gr-zero
      the-enat.simps)
next
   fix n
   assume a0: enat n ≤ nlength σ
   assume a1: f (ndropn n σ)
   assume a2:  ∀ na. na ≤ the-enat (min (enat (n − Suc 0)) (epred (nlength σ))) ∧ na ≤ n ∧
                  enat na ≤ nlength σ ⟶ w (NNil (nnth σ na))
   show  ∃ k. enat k ≤ nlength σ ∧ f (ndropn k σ) ∧ (∀ j<k. w (NNil (nnth σ j)))
   proof −
   have 1: enat n ≤ nlength σ ∧ f (ndropn n σ)
    using a0 a1 by auto
   have 2: (∀ na. na ≤ the-enat (min (enat (n − Suc 0)) (epred (nlength σ))) ∧ na ≤ n ∧
                enat na ≤ nlength σ ⟶ w (NNil (nnth σ na))) ⟶
           (∀ j<n. w (NNil (nnth σ j)))
     by (auto simp add: min-def)
       (simp add: a0 less-imp-le order-less-le-subst2,
        metis One-nat-def a0 epred-enat epred-min min.absorb-iff2)
   have 3: (∀ j<n. w (NNil (nnth σ j)))
     using 2 a2 by blast
   show ?thesis
   using 1 3 by blast
  qed
qed

lemma StateUntilEqvWPrevChop:
⊢ (init w) 𝒰 f = (wprev (□ (init w)))⌢f
using StateUntilEqvWPrevChopsem Valid-def by blast
```

lemma *UntilChopDist*:
$\vdash$ (*init w*) $\mathcal{U}$ (*g*$\frown$*h*) = ((*init w*) $\mathcal{U}$ *g*) $\frown$*h*
using *StateUntilEqvWPrevChop*[*of w*]
by (*metis SChopAssoc inteq-reflection*)

lemma *PiEmptysem*:
$\sigma \models$ (*init w*) Π *empty* = (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*))))
proof −
have *1*: ($\sigma \models$ (*init w*) Π *empty*) =
($\exists$ *x*$\in$*nset* σ. *w* (*NNil x*)) $\wedge$ *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) σ) = *0*)
by (*simp add*: *Pistate itl-defs zero-enat-def*)
have *2*: ($\exists$ *x*$\in$*nset* σ. *w* (*NNil x*)) $\wedge$ *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) σ) = *0*) =
($\exists$ *k* $\leq$ *nlength* σ . (λ*y*. *w* (*NNil y*)) (*nnth* σ *k*) $\wedge$
($\forall$ *j*. *j*<*k* $\longrightarrow$ $\neg$ (λ*y*. *w* (*NNil y*)) (*nnth* σ *j*)) $\wedge$
($\forall$ *j* $\leq$ *nlength* σ. *k*<*j* $\longrightarrow$ $\neg$ (λ*y*. *w* (*NNil y*)) (*nnth* σ *j*)))
by (*simp add*: *nfilter-nlength-zero-conv-2*)
have *3*: ($\exists$ *k* $\leq$ *nlength* σ . (λ*y*. *w* (*NNil y*)) (*nnth* σ *k*) $\wedge$
($\forall$ *j*. *j*<*k* $\longrightarrow$ $\neg$ (λ*y*. *w* (*NNil y*)) (*nnth* σ *j*)) $\wedge$
($\forall$ *j* $\leq$ *nlength* σ. *k*<*j* $\longrightarrow$ $\neg$ (λ*y*. *w* (*NNil y*)) (*nnth* σ *j*))) =
($\exists$ *k*$\leq$*nlength* σ.
w (*NNil* (*nnth* σ *k*)) $\wedge$
(*LIFT*(*wnext* ($\Box$ (*init* ($\neg$ *w*))))) (*ndropn k* σ) $\wedge$
($\forall$ *j*<*k*. $\neg$ *w* (*NNil* (*nnth* σ *j*)))) (**is** *?lhs* =*?rhs*)
proof
assume *a*: *?lhs*
show *?rhs* **using** *a wnextboxnotstatesem* **by** *auto*
next
assume *b*: *?rhs*
show *?lhs*
proof −
obtain *k* **where** *1*:
enat k $\leq$ *nlength* σ $\wedge$ *w* (*NNil* (*nnth* σ *k*)) $\wedge$
(*LIFT*(*wnext* ($\Box$ (*init* ($\neg$ *w*))))) (*ndropn k* σ) $\wedge$ ($\forall$ *j*<*k*. $\neg$ *w* (*NNil* (*nnth* σ *j*)))
using *b* **by** *auto*
have *2*: *enat k* $\leq$ *nlength* σ $\wedge$ *w* (*NNil* (*nnth* σ *k*)) $\wedge$ ($\forall$ *j*<*k*. $\neg$ *w* (*NNil* (*nnth* σ *j*))) $\wedge$
($\forall$ *j*. *enat j* $\leq$ *nlength* σ $\longrightarrow$ *k* < *j* $\longrightarrow$ $\neg$ *w* (*NNil* (*nnth* σ *j*)))
using *1 wnextboxnotstatesem*[*of k* σ *w*]
proof −
have $\forall$ *x0*. (*x0* < *k* $\longrightarrow$ $\neg$ *w* (*NNil* (*nnth* σ *x0*))) = ($\neg$ *x0* < *k* $\vee$ $\neg$ *w* (*NNil* (*nnth* σ *x0*)))
by *meson*
then have *f1*: *enat k* $\leq$ *nlength* σ $\wedge$ *w* (*NNil* (*nnth* σ *k*)) $\wedge$
(*LIFT*(*wnext* ($\Box$ (*init* ($\neg$ *w*))))) (*ndropn k* σ) $\wedge$ ($\forall$ *n*. $\neg$ *n* < *k* $\vee$ $\neg$ *w* (*NNil* (*nnth* σ *n*)))
by (*metis* ‹*enat k* $\leq$ *nlength* σ $\wedge$ *w* (*NNil* (*nnth* σ *k*)) $\wedge$
(*LIFT*(*wnext* ($\Box$ (*init* ($\neg$ *w*))))) (*ndropn k* σ) $\wedge$ ($\forall$ *j*<*k*. $\neg$ *w* (*NNil* (*nnth* σ *j*)))›)
have (*enat k* $\leq$ *nlength* σ $\longrightarrow$ ($\forall$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$ *k* < *n* $\longrightarrow$ $\neg$ *w* (*NNil* (*nnth* σ *n*))) =
(*LIFT*(*wnext* ($\Box$ (*init* ($\neg$ *w*))))) (*ndropn k* σ)) =
($\neg$ *enat k* $\leq$ *nlength* σ $\vee$ ($\forall$ *n*. ($\neg$ *enat n* $\leq$ *nlength* σ $\vee$ $\neg$ *k* < *n*) $\vee$ $\neg$ *w* (*NNil* (*nnth* σ *n*))) =
(*LIFT*(*wnext* ($\Box$ (*init* ($\neg$ *w*))))) (*ndropn k* σ))
by *blast*

then have ¬ *enat k* ≤ *nlength σ* ∨ (∀ *n*. (¬ *enat n* ≤ *nlength σ* ∨ ¬ *k* < *n*) ∨ ¬ *w* (*NNil* (*nnth σ n*))) =
(*LIFT*(*wnext* (□ (*init* (¬ *w*))))) (*ndropn k σ*)
by (*metis* ‹*enat k* ≤ *nlength σ* ⟹ (∀ *j*. *enat j* ≤ *nlength σ* ⟶ *k* < *j* ⟶ ¬ *w* (*NNil* (*nnth σ j*))) =
(*LIFT*(*wnext* (□ (*init* (¬ *w*))))) (*ndropn k σ*)›)
then show *?thesis* **using** *f1* **by** *presburger*
qed
show *?thesis* **using** *2* **by** *blast*
qed
qed
have *4*: (∃ *k*≤*nlength σ*.
w (*NNil* (*nnth σ k*)) ∧
(*LIFT*(*wnext* (□ (*init* (¬ *w*))))) (*ndropn k σ*) ∧
(∀ *j*<*k*. ¬ *w* (*NNil* (*nnth σ j*)))) =
(*σ* ⊨ (*init* (¬ *w*)) 𝒰 ((*init w*) ∧ *wnext* (□ (*init* (¬ *w*)))))
by (*simp add*: *NotStateUntilStateAndsem*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *PiEmpty*:
⊢ (*init w*) Π *empty* = (*init* (¬ *w*)) 𝒰 ((*init w*) ∧ *wnext* (□ (*init* (¬ *w*))))
using *PiEmptysem Valid-def* **by** *blast*

lemma *StatePiBoxStatesem*:
σ ⊨ (*init w*) Π *f* = (*init w*) Π (*f* ∧ □ (*init w*))
proof −
have *1*: (*σ* ⊨ (*init w*) Π *f*) =
((∃ *x* ∈ *nset σ*. *w* (*NNil x*)) ∧ ((*nfilter* (λ*y*. *w* (*NNil y*)) *σ*) ⊨ *f*))
by (*metis Pistate*)
have *2*: (*σ* ⊨ (*init w*) Π (*f* ∧ □ (*init w*))) =
((∃ *x* ∈ *nset σ*. *w* (*NNil x*)) ∧ ((*nfilter* (λ*y*. *w* (*NNil y*)) *σ*) ⊨ *f* ∧ □ (*init w*)))
by (*metis Pistate*)
have *3*: ((*nfilter* (λ*y*. *w* (*NNil y*)) *σ*) ⊨ *f* ∧ □ (*init w*))
= (*f* (*nfilter* (λ*y*. *w* (*NNil y*)) *σ*) ∧
(∀ *n*≤*nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) *σ*). *w* (*NNil* (*nnth* (*nfilter* (λ*y*. *w* (*NNil y*)) *σ*) *n*))))
by (*simp add*: *itl-defs ndropn-nfirst*)
have *4*: ((∃ *x* ∈ *nset σ*. *w* (*NNil x*)) ∧ (*f* (*nfilter* (λ*y*. *w* (*NNil y*)) *σ*) ∧
(∀ *n*≤*nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) *σ*). *w* (*NNil* (*nnth* (*nfilter* (λ*y*. *w* (*NNil y*)) *σ*) *n*))))) =
((∃ *x* ∈ *nset σ*. *w* (*NNil x*)) ∧ *f* (*nfilter* (λ*y*. *w* (*NNil y*)) *σ*))
by (*meson nkfilter-nnth-aa*)
show *?thesis* **using** *1 2 3 4* **by** *auto*
qed

lemma *StatePiBoxState*:
⊢ (*init w*) Π *f* = (*init w*) Π (*f* ∧ □ (*init w*))
using *StatePiBoxStatesem Valid-def* **by** *blast*

lemma *StatePiUntil1*:
⊢ ((*init* (¬ *w*)) 𝒰 ((*init w*) ∧ (*init w*) Π *f*)) =

$(wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ )$
using *StateUntilEqvWPrevChop* **by** *blast*

lemma *StatePiUntilsem2*:
$(\sigma \models (wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models ((wprev\ (\Box\ (init\ (\neg\ w))))\frown((init\ w)\ \wedge\ empty))\ \frown\ ((init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ )\ )$
by (*auto simp add*: *schop-defs init-defs empty-defs zero-enat-def ndropn-nfirst*)
(*metis* (*no-types*, *lifting*) *add.right-neutral dual-order.refl enat.simps*(*3*) *enat-add-sub-same min.orderE min.orderE nfinite-ntaken nnth-nlast ntaken-nlast ntaken-nlength the-enat.simps zero-enat-def*)

lemma *StatePiUntil2*:
$\vdash (wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ ) =$
$(\ ((wprev\ (\Box\ (init\ (\neg\ w))))\frown((init\ w)\ \wedge\ empty))\ \frown\ ((init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ )\ )$
by (*simp add*: *StatePiUntilsem2 Valid-def*)

lemma *StatePiUntil3*:
$\vdash ((wprev\ (\Box\ (init\ (\neg\ w))))\frown((init\ w)\ \wedge\ empty)) =$
$(\ ((init\ (\neg\ w))\ \mathcal{U}\ ((init\ w)\ \wedge\ wnext\ (\Box\ (init\ (\neg\ w)))))\ \frown\ ((init\ w)\ \wedge\ empty))$
proof −
have *1*: $\vdash (wprev\ (\Box\ (init\ (\neg\ w))))\frown((init\ w)\ \wedge\ empty) =$
$(init\ (\neg\ w))\ \mathcal{U}\ ((init\ w)\ \wedge\ empty)$
by (*meson Prop11 StateUntilEqvWPrevChop*)
have *2*: $\vdash ((init\ w)\ \wedge\ empty) = ((init\ w)\ \wedge\ wnext\ (\Box\ (init\ (\neg\ w))))\frown((init\ w)\ \wedge\ empty)$
by (*auto simp add*: *Valid-def itl-defs zero-enat-def ndropn-nfirst*)
(*metis ndropn-0 ndropn-nfirst* ,
metis add.right-neutral add-diff-inverse-nat enat.simps(*3*) *enat-add-sub-same enat-ord-simps*(*2*) *iless-Suc-eq less-eqE min.absorb2 min-def ntaken-all one-eSuc one-enat-def order-refl plus-1-eq-Suc zero-enat-def zero-le*)
show *?thesis* **by** (*metis 1 2 UntilChopDist inteq-reflection*)
qed

lemma *StatePiUntilsem4*:
$(\sigma \models ((init\ (\neg\ w))\ \mathcal{U}\ ((init\ w)\ \wedge\ wnext\ (\Box\ (init\ (\neg\ w)))))\ \frown\ ((init\ w)\ \wedge\ empty)) =$
$(\sigma \models ((init\ w)\ \Pi\ empty)\frown((init\ w)\ \wedge\ empty))$
by (*metis PiEmpty inteq-reflection*)

lemma *StatePiUntil4*:
$\vdash ((init\ (\neg\ w))\ \mathcal{U}\ ((init\ w)\ \wedge\ wnext\ (\Box\ (init\ (\neg\ w)))))\ \frown\ ((init\ w)\ \wedge\ empty) =$
$(\ ((init\ w)\ \Pi\ empty)\frown((init\ w)\ \wedge\ empty))$
by (*simp add*: *StatePiUntilsem4 Valid-def*)

lemma *StatePiUntilsem*:
$\sigma \models (init\ w)\ \Pi\ f = (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ )$
proof −
have *2*: $(\sigma \models (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w)\ \wedge\ (init\ w)\ \Pi\ f\ ))$
using *StateUntilEqvWPrevChopsem*[*of LIFT*$(\neg\ w)$ *LIFT*$((init\ w)\ \wedge\ (init\ w)\ \Pi\ f)\ \sigma$]
by *simp*

have *7*: $(\sigma \models (wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (((init\ w)\ \Pi\ empty)\frown\ ((init\ w) \wedge empty))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ ))$
by (*metis PiEmpty StatePiUntil2 StatePiUntil3 inteq-reflection*)
have *8*: $(\sigma \models (((init\ w)\ \Pi\ empty)\frown\ ((init\ w) \wedge empty))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (((init\ w)\ \Pi\ empty))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ ))$
by (*auto simp add*: *schop-defs init-defs empty-defs zero-enat-def ndropn-nfirst*)
(*metis* (*no-types*, *opaque-lifting*) *enat-0-iff*(*2*) *min.absorb-iff1 ndropn-all ndropn-nlength nle-le nlength-NNil ntaken-nlast ntaken-nlength ntaken-ntaken*)
have *9*: $(\sigma \models (((init\ w)\ \Pi\ empty))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (init\ w)\ \Pi\ (empty\frown f))$
using *PiSChopDistsema PiSChopDistsemb* **by** *blast*
have *10*: $(\sigma \models (init\ w)\ \Pi\ (empty\frown f)) = (\sigma \models (init\ w)\ \Pi\ f)$
by (*metis EmptySChop inteq-reflection*)
show *?thesis*
by (*simp add*: *10 2 7 8 9*)
qed

lemma *StatePiUntil*:
$\vdash (init\ w)\ \Pi\ f = (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )$
using *StatePiUntilsem* **by** *blast*

lemma *StateAndPiEmpty*:
$\vdash\ ((init\ w) \wedge (init\ w)\ \Pi\ empty) = (w \wedge empty) \frown (wnext\ (\Box\ (init\ (\neg\ w))))$
proof −
have *1*: $\vdash\ ((init\ w) \wedge (init\ w)\ \Pi\ empty) =$
$((init\ w) \wedge (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w)))))\ )$
using *PiEmpty* **by** *fastforce*
have *2*: $\vdash ((init\ w) \wedge (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w)))))\ )$
$= ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w)))))$
by (*auto simp add*: *until-d-def Valid-def init-defs ndropn-nfirst*)
(*metis ndropn-0 ndropn-nfirst neq0-conv*,
metis gr-implies-not-zero ndropn-0 ndropn-nfirst zero-enat-def zero-le)
have *3*: $\vdash ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w))))) = (w \wedge empty) \frown (wnext\ (\Box\ (init\ (\neg\ w))))$
proof −
have $\bigwedge p\ pa.\ \vdash((p::'a\ nellist \Rightarrow bool) \wedge empty) \frown pa = (init\ p \wedge pa)$
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop inteq-reflection*)
then show *?thesis*
by (*simp add*: *Prop11*)
qed
show *?thesis*
by (*metis 1 2 3 inteq-reflection*)
qed

lemma *PiFPowerExpandsem*:
$(\sigma \models (\exists\, k.\ (init\ w)\ \Pi\ (fpower\ f\ k))) =$
$(\sigma \models (init\ w)\ \Pi\ empty \vee (\exists\, k.\ (init\ w)\ \Pi\ (f\frown(fpower\ f\ (k)))))$
proof −
have *1*: $(\sigma \models (\exists\, k.\ (init\ w)\ \Pi\ (fpower\ f\ k))) =$
$(\exists\ k.\ (\sigma \models (init\ w)\ \Pi\ (fpower\ f\ k)))$
by *simp*

have *2*: $(\exists\ k.\ (\sigma \models (init\ w)\ \Pi\ (fpower\ f\ k))) =$
$(\ (\sigma \models (init\ w)\ \Pi\ (fpower\ f\ 0)) \vee \quad (\exists\ k.\ 1 \leq k \wedge (\sigma \models (init\ w)\ \Pi\ (fpower\ f\ (k)))))$
by (*metis One-nat-def diff-Suc-1 le-SucE le-add1 plus-1-eq-Suc*)
have *3*: $(\sigma \models (init\ w)\ \Pi\ (fpower\ f\ 0)) = (\sigma \models (init\ w)\ \Pi\ empty)$
by (*simp add*: *itl-def fpower-d-def*)
have *4*: $(\exists\ k.\ 1 \leq k \wedge (\sigma \models (init\ w)\ \Pi\ (fpower\ f\ (k)))) =$
$(\exists\ k.\ (\sigma \models (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))))$
by (*metis le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have *5*: $(\exists\ k.\ (\sigma \models (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)))) =$
$(\sigma \models (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))))$
by *simp*
have *6*: $((\sigma \models (init\ w)\ \Pi\ empty) \vee (\sigma \models (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))))) =$
$(\sigma \models (init\ w)\ \Pi\ empty \vee (\exists k.\ (init\ w)\ \Pi\ (f \frown fpower\ f\ (k))))$
unfolding *fpower-d-def* **by** (*auto simp add*: *schop-d-def*)
show *?thesis* **using** *1 2 3 4 5 6* **by** *blast*
qed

lemma *PiFPowerExpandsem1*:
$\forall\ \sigma.\ \sigma \models\ (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$(\ (init\ w)\ \Pi\ empty\ \ \vee (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))\ \ )\ )$
proof
fix σ
show $\sigma \models\ (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$(\ (init\ w)\ \Pi\ empty\ \ \vee (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))\ \ )\ )$
proof −
have *1*: $(\sigma \models\ (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$(\ (init\ w)\ \Pi\ empty\ \ \vee (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))\ \ )\ ))$
$= (\ (\sigma \models\ (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ k))) =$
$(\sigma \models (init\ w)\ \Pi\ empty\ \ \vee (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))\ \ )\ )\ )$

by *auto*
have *2*: $(\ (\sigma \models\ (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ k))) =$
$(\sigma \models (init\ w)\ \Pi\ empty\ \ \vee (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))\ \ )\ )\ )$
using *PiFPowerExpandsem*[*of w f* σ]
unfolding *fpower-d-def* **by** (*auto simp add*: *schop-d-def*)
show *?thesis* **using** *1 2* **by** *blast*
qed
qed

lemma *PiFPowerExpand*:
$\vdash\ (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$(\ (init\ w)\ \Pi\ empty\ \ \vee (\exists k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))\ \ )\ )$
using *PiFPowerExpandsem1*[*of w f*] **by** (*auto simp add*: *Valid-def PiFPowerExpandsem1*)

lemma *exists-expand-sem*:
$(\sigma \models (\exists k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ k))) =$
$((\sigma \models (fpower\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ 0)) \vee$
$(\sigma \models (\exists k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ (Suc\ k)))))$
by (*metis* (*no-types*, *lifting*) *not0-implies-Suc unl-Rex*)

lemma *exists-expand*:
$\vdash$ $(\exists\, k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ k)) =$
$((fpower\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ 0) \vee (\exists\, k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ (Suc\ k))))$
using *exists-expand-sem Valid-def* **by** *fastforce*

TruePiEqv

lemma *TruePiEqvsem*:
$\sigma \models \#True\ \Pi\ f = f$
by (*simp add*: *pi-d-def pifilt-true*) (*metis zero-enat-def zero-le*)

lemma *TruePiEqv*:
$\vdash (\#True)\ \Pi\ f = f$
using *TruePiEqvsem* **by** (*auto simp add*: *Valid-def*)

BoxImpEqvPi

lemma *BoxImpEqvPi*:
$\vdash \Box\ f \longrightarrow g = f\ \Pi\ g$
proof (*simp add*: *Valid-def itl-defs pi-d-def pifilt-def sfxfilt-def*)
show $\forall\, w.\ (\forall\, n.\ enat\ n \leq nlength\ w \longrightarrow f\ (ndropn\ n\ w)) \longrightarrow$
$g\ w = ((\exists\, i.\ enat\ i \leq nlength\ w \wedge f\ (ndropn\ i\ w)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w))))$
proof
fix *w*
show $(\forall\, n.\ enat\ n \leq nlength\ w \longrightarrow f\ (ndropn\ n\ w)) \longrightarrow$
$g\ w = ((\exists\, i.\ enat\ i \leq nlength\ w \wedge f\ (ndropn\ i\ w)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w))))$
proof
assume *a0*: $\forall\, n.\ enat\ n \leq nlength\ w \longrightarrow f\ (ndropn\ n\ w)$
show $g\ w = ((\exists\, i.\ enat\ i \leq nlength\ w \wedge f\ (ndropn\ i\ w)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w))))$
proof −
have *1*: $(nfilter\ f\ (ndropns\ w)) = (ndropns\ w)$
by (*metis* (*mono-tags*, *lifting*) *a0 ile0-eq in-nset-ndropns le-cases nfilter-id-conv zero-enat-def*)
have *2*: $w = (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w)))$
by (*simp add*: *1 nmap-first-ndropns*)
show *?thesis* **by** (*metis 1 a0 nmap-first-ndropns zero-enat-def zero-le*)
qed
qed
qed
qed

PiEqvDiamondUPi

lemma *PiEqvDiamondUPi*:
$\vdash f\ \Pi\ g = (\ \Diamond\ f \wedge f\ \Pi^u\ g)$
by (*simp add*: *Valid-def upi-d-def itl-defs pi-d-def*,*blast*)

PiEqvUntilPi

lemma *PiEqvUntilPi*:
$\vdash$ (*init w*) Π *g* = (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) Π *g*)
by (*metis StatePiUntil UntilUntil int-eq*)

UPiEqvBoxOrPi

lemma *UPiEqvBoxOrPi*:
$\vdash$ *f* Π^u *g* = ($\Box$ ($\neg$ *f*) $\vee$ *f* Π *g*)
by (*simp add*: *Valid-def upi-d-def itl-defs pi-d-def*,*blast*)

3.9.4 Theorems

lemma *UPiImpRule*:
assumes $\vdash$ *g1* $\longrightarrow$ *g2*
shows $\vdash$ *f* Π^u *g1* $\longrightarrow$ *f* Π^u *g2*
using *assms*
by (*meson MP PiK PiN*)

lemma *UPiEqvRule*:
assumes $\vdash$ *g1* = *g2*
shows $\vdash$ *f* Π^u *g1* = *f* Π^u *g2*
proof −
have *1*: $\vdash$ *g1* $\longrightarrow$ *g2*
using *assms* **by** (*simp add*: *int-iffD1*)
have *2*: $\vdash$ *f* Π^u *g1* $\longrightarrow$ *f* Π^u *g2*
using *1 UPiImpRule* **by** *blast*
have *3*: $\vdash$ *g2* $\longrightarrow$ *g1*
using *assms* **by** (*simp add*: *int-iffD2*)
have *4*: $\vdash$ *f* Π^u *g2* $\longrightarrow$ *f* Π^u *g1*
using *3 UPiImpRule* **by** *blast*
from *3 4* **show** *?thesis*
by (*simp add*: *2 int-iffI*)
qed

lemma *PiEqvNotUPiNot*:
$\vdash$ *f* Π *g* = ($\neg$ (*f* Π^u ($\neg$ *g*)))
by (*simp add*: *upi-d-def*)

lemma *NotPiEqvNotUPi*:
$\vdash$ *f* Π ($\neg$ *g*) = ($\neg$ (*f* Π^u *g*))
by (*simp add*: *upi-d-def*)

lemma *UPiEqvNotPiNot*:
$\vdash$ *f* Π^u *g* = ($\neg$ (*f* Π ($\neg$ *g*)))
by (*simp add*: *upi-d-def*)

lemma *NotUPiEqvNotPi*:
$\vdash$ *f* Π^u ($\neg$*g*) = ($\neg$ (*f* Π *g*))
by (*simp add*: *upi-d-def*)

lemma *PiImpRule*:
assumes $\vdash$ *g1* $\longrightarrow$ *g2*
shows $\vdash$ *f* Π *g1* $\longrightarrow$ *f* Π *g2*
proof −
have *1*: $\vdash \neg$ *g2* $\longrightarrow \neg$ *g1*
by (*simp add*: *assms*)
have *2*: $\vdash$ *f* Π^u ($\neg$ *g2*) $\longrightarrow$ *f* Π^u ($\neg$*g1*)
using *1 UPiImpRule* **by** *blast*
have *3*: $\vdash \neg$(*f* Π^u ($\neg$*g1*)) $\longrightarrow \neg$(*f* Π^u ($\neg$*g2*))
using *2* **by** *fastforce*
from *3* **show** *?thesis* **using** *PiEqvNotUPiNot* **by** *fastforce*
qed

lemma *PiEqvRule*:
assumes $\vdash$ *g1* = *g2*
shows $\vdash$ *f* Π *g1* = *f* Π *g2*
proof −
have *1*: $\vdash$ *g1* $\longrightarrow$ *g2*
using *assms* **by** (*simp add*: *int-iffD1*)
have *2*: $\vdash$ *f* Π *g1* $\longrightarrow$ *f* Π *g2*
using *1 PiImpRule* **by** *blast*
have *3*: $\vdash$ *g2* $\longrightarrow$ *g1*
using *assms* **by** (*simp add*: *int-iffD2*)
have *4*:$\vdash$ *f* Π *g2* $\longrightarrow$ *f* Π *g1*
using *3 PiImpRule* **by** *blast*
from *2 4* **show** *?thesis* **by** (*simp add*: *int-iffI*)
qed

lemma *UPiAndPiImpPiAnd*:
$\vdash$ *f1* Π^u *f* $\wedge$ *f1* Π ($\neg$ *g*) $\longrightarrow$ *f1* Π (*f* $\wedge \neg$*g*)
proof −
have *1*: $\vdash$ ($\neg$(*f* $\longrightarrow$ *g*)) = (*f* $\wedge \neg$*g*)
by *fastforce*
have *2*: $\vdash$ ($\neg$ (*f1* Π^u (*f* $\longrightarrow$*g*))) = *f1* Π ($\neg$(*f* $\longrightarrow$ *g*))
by (*simp add*: *NotPiEqvNotUPi int-iffD1 int-iffD2 int-iffI*)
have *3*: $\vdash$ $\neg$(*f1* Π^u *f* $\longrightarrow$ *f1* Π^u *g*) $\longrightarrow \neg$ (*f1* Π^u (*f* $\longrightarrow$*g*))
by (*simp add*: *PiK*)
have *4*: $\vdash$ ($\neg$(*f1* Π^u *f* $\longrightarrow$ *f1* Π^u *g*)) = (*f1* Π^u *f* $\wedge$ *f1* Π ($\neg$ *g*))
using *NotPiEqvNotUPi*[*of f1 g*] **by** *fastforce*
have *5*:$\vdash$ *f1* Π ($\neg$(*f* $\longrightarrow$ *g*)) = *f1* Π (*f* $\wedge \neg$*g*)
using *1* **by** (*simp add*: *PiEqvRule*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *UPiAndPiImpPiAndA*:
$\vdash$ *f1* Π^u *f* $\wedge$ *f1* Π *g* $\longrightarrow$ *f1* Π (*f* $\wedge$ *g*)
using *UPiAndPiImpPiAnd*[*of f1 f LIFT*($\neg$*g*)] **by** *fastforce*

lemma *PiAndPiImpPiAnd*:

$\vdash$ *f1* Π *f* $\wedge$ *f1* Π *g* $\longrightarrow$ *f1* Π (*f* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *f1* Π^u *f* $\wedge$ *f1* Π *g* $\longrightarrow$ *f1* Π (*f* $\wedge$ *g*)
using *UPiAndPiImpPiAndA* **by** *fastforce*
have *2*: $\vdash$ *f1* Π *f* $\longrightarrow$ *f1* Π^u *f*
using *PiDc* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *PiAnd*:
$\vdash$ *f* Π (*g1* $\wedge$ *g2*) = (*f* Π *g1* $\wedge$ *f* Π *g2*)
proof −
have *1*: $\vdash$ *f* Π (*g1* $\wedge$ *g2*) $\longrightarrow$ *f* Π *g1*
by (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)
have *2*: $\vdash$ *f* Π (*g1* $\wedge$ *g2*) $\longrightarrow$ *f* Π *g2*
by (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)
have *3*: $\vdash$ *f* Π (*g1* $\wedge$ *g2*) $\longrightarrow$ *f* Π *g1* $\wedge$ *f* Π *g2*
using *1 2* **by** *fastforce*
have *4*: $\vdash$ *f* Π *g1* $\wedge$ *f* Π *g2* $\longrightarrow$ *f* Π (*g1* $\wedge$ *g2*)
by (*simp add*: *PiAndPiImpPiAnd*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *UPiAnd*:
$\vdash$ *f* Π^u (*g1* $\wedge$ *g2*) = (*f* Π^u *g1* $\wedge$ *f* Π^u *g2*)
proof −
have *1*: $\vdash$ *f* Π ($\neg$*g1* $\vee$ $\neg$ *g2*) = (*f* Π ($\neg$*g1*) $\vee$ *f* Π ($\neg$ *g2*))
by (*simp add*: *PiOr*)
have *2*: $\vdash$ ($\neg$(*f* Π ($\neg$*g1* $\vee$ $\neg$ *g2*))) = ($\neg$(*f* Π ($\neg$*g1*) $\vee$ *f* Π ($\neg$ *g2*)))
using *1* **by** *fastforce*
have *3*: $\vdash$ ($\neg$(*f* Π ($\neg$*g1* $\vee$ $\neg$ *g2*))) = *f* Π^u ($\neg$($\neg$*g1* $\vee$ $\neg$ *g2*))
by (*meson NotUPiEqvNotPi Prop11*)
have *4*: $\vdash$ ($\neg$($\neg$*g1* $\vee$ $\neg$ *g2*)) = (*g1* $\wedge$ *g2*)
by *fastforce*
have *5*: $\vdash$ *f* Π^u ($\neg$($\neg$*g1* $\vee$ $\neg$ *g2*)) = *f* Π^u (*g1* $\wedge$ *g2*)
using *4* **by** (*simp add*: *UPiEqvRule*)
have *6*: $\vdash$ ($\neg$(*f* Π ($\neg$*g1*) $\vee$ *f* Π ($\neg$ *g2*))) = ($\neg$(*f* Π ($\neg$*g1*)) $\wedge$ $\neg$(*f* Π($\neg$*g2*)))
by *fastforce*
have *7*: $\vdash\neg$(*f* Π ($\neg$*g1*)) = *f* Π^u *g1*
by (*simp add*: *NotPiEqvNotUPi*)
have *8*: $\vdash$ $\neg$(*f* Π ($\neg$*g2*)) = *f* Π^u *g2*
by (*simp add*: *NotPiEqvNotUPi*)
have *9*: $\vdash$ ($\neg$(*f* Π ($\neg$*g1*)) $\wedge$ $\neg$(*f* Π($\neg$*g2*))) = (*f* Π^u *g1* $\wedge$ *f* Π^u *g2*)
using *6 7 8* **by** *fastforce*
show *?thesis* **by** (*metis 2 3 5 6 9 inteq-reflection*)
qed

lemma *UPiOr*:
$\vdash$ *f* Π^u (*g1* $\vee$ *g2*) = (*f* Π^u *g1* $\vee$ *f* Π^u *g2*)
proof −

have *1*: $\vdash f\ \Pi\ (\neg g1 \wedge \neg g2) = (f\ \Pi\ (\neg g1) \wedge f\ \Pi\ (\neg g2))$
by (*simp add*: *PiAnd*)
have *2*: $\vdash (\neg(f\ \Pi\ (\neg g1 \wedge \neg g2)))\ = (\neg(f\ \Pi\ (\neg g1) \wedge f\ \Pi\ (\neg g2)))$
using *1* **by** *fastforce*
have *3*: $\vdash (\neg(f\ \Pi\ (\neg g1 \wedge \neg g2))) = f\ \Pi^u\ (\neg(\neg g1 \wedge \neg\ g2))$
by (*meson NotUPiEqvNotPi Prop11*)
have *4*: $\vdash (\neg(\neg g1 \wedge \neg\ g2)) = (g1 \vee g2)$
by *fastforce*
have *5*: $\vdash f\ \Pi^u\ (\neg(\neg g1 \wedge \neg\ g2)) = f\ \Pi^u\ (g1 \vee g2)$
using *4 UPiEqvRule* **by** *blast*
have *6*: $\vdash (\neg(f\ \Pi\ (\neg g1) \wedge f\ \Pi\ (\neg g2))) = (\neg(f\ \Pi\ (\neg g1)) \vee \neg(f\ \Pi\ (\neg g2)))$
by *fastforce*
have *7*: $\vdash (\neg(f\ \Pi\ (\neg g1))) = f\ \Pi^u\ g1$
by (*simp add*: *upi-d-def*)
have *8*: $\vdash (\neg(f\ \Pi\ (\neg g2))) = f\ \Pi^u\ g2$
by (*simp add*: *upi-d-def*)
have *9*: $\vdash (\neg(f\ \Pi\ (\neg g1)) \vee \neg(f\ \Pi\ (\neg g2))) = (\ f\ \Pi^u\ g1 \vee f\ \Pi^u\ g2)$
using *7 8* **by** *fastforce*
show *?thesis*
by (*metis 2 3 4 6 9 inteq-reflection*)
qed

lemma *UpiAndImp*:
$\vdash f\ \Pi^u\ (g1 \longrightarrow g2) \wedge f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2$
proof −
have *2*: $\vdash f\ \Pi^u\ ((\neg g2) \longrightarrow (\neg g1)) \longrightarrow (\ f\ \Pi^u\ (\neg\ g2) \longrightarrow f\ \Pi^u\ (\neg\ g1)\ )$
using *PiK* **by** *blast*
have *3*: $\vdash (f\ \Pi^u\ (\neg g2) \longrightarrow f\ \Pi^u\ (\neg g1)) = ((\neg\ (f\ \Pi^u\ (\neg g1))) \longrightarrow (\neg(f\ \Pi^u\ (\neg g2))))$
by *auto*
have *4*: $\vdash (\neg\ (f\ \Pi^u\ (\neg g2))) = f\ \Pi\ g2$
by (*simp add*: *upi-d-def*)
have *5*: $\vdash (\neg\ (f\ \Pi^u\ (\neg g1))) = f\ \Pi\ g1$
by (*simp add*: *upi-d-def*)
have *6*: $\vdash f\ \Pi^u\ ((\neg g2) \longrightarrow (\neg g1)) = f\ \Pi^u\ (g1 \longrightarrow g2)$
by *simp*
have *7*: $\vdash (f\ \Pi^u\ (g1 \longrightarrow g2) \wedge f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2) =$
$(f\ \Pi^u\ (g1 \longrightarrow g2)\ \longrightarrow (f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2))$
by *auto*
show *?thesis*
using *2 4 5* **by** *fastforce*
qed

lemma *BoxImpUPiBox*:
$\vdash \Box\ (init\ w) \longrightarrow f\ \Pi^u\ (\Box\ (init\ w))$
proof −
have *1*: $\vdash f\ \Pi\ (\Diamond\ (init\ (\neg w))) \longrightarrow \Diamond\ (init\ (\neg w))$
by (*simp add*: *PiDiamondImp*)
have *2*: $\vdash \neg\ \Diamond\ (init\ (\neg w)) \longrightarrow \neg\ (f\ \Pi\ (\Diamond\ (init\ (\neg w))))$
using *1* **by** *auto*

have *3*: $\vdash (\neg\ \Diamond\ (init\ (\neg w))) = \Box\ (init\ w)$
by (*metis 2 Initprop(2) Prop10 always-d-def inteq-reflection*)
have *4*: $\vdash (\neg\ (f\ \Pi\ (\Diamond\ (init\ (\neg w))))) = f\ \Pi^u\ \ (\Box\ (init\ w))$
by (*simp add*: *upi-d-def*)
(*metis 3 int-simps(4) inteq-reflection*)
show *?thesis*
using *2 3 4* **by** *fastforce*
qed

lemma *WPrevPi*:
$\vdash (init\ w)\ \Pi\ f = (wprev\ (\Box\ (init\ (\neg\ w))))\frown\ ((init\ w) \wedge (init\ w)\ \Pi\ f)$
using *StatePiUntil StatePiUntil1* **by** *fastforce*

lemma *EmptyAndSChopAndMoreEqvAndSChop*:
$\vdash (w \wedge empty)\frown(f \wedge more) = ((w \wedge empty)\frown f \wedge more)$
proof −
have *1*: $\vdash (w \wedge empty)\frown(f \wedge more) \longrightarrow (w \wedge empty)\frown f$
by (*simp add*: *SChopAndA*)
have *2*: $\vdash (w \wedge empty)\frown(f \wedge more) \longrightarrow more$
by (*meson SChopAndB SChopMoreImpMore lift-imp-trans*)
have *3*: $\vdash((w \wedge empty)\frown f \wedge more) \longrightarrow (w \wedge empty)\frown(f \wedge more)$
by (*metis* (*no-types*, *opaque-lifting*) *InitAndEmptyEqvAndEmpty Prop11 Prop12 StateAndEmptySChop*
int-simps(1) inteq-reflection)
show *?thesis*
by (*simp add*: *1 2 3 Prop12 int-iffI*)
qed

lemma *PiInfImpInf*:
$\vdash f\ \Pi\ inf \longrightarrow inf$
unfolding *Valid-def pi-d-def init-defs infinite-defs*
by *auto*
(*metis enat-ile nfinite-conv-nlength-enat pifilt-nlength-bound*)

lemma *DiamondWPrevBoxSChop*:
$\vdash \Diamond\ (init\ w) = (wprev\ (\Box\ (init\ (\neg\ w))))\frown\ (init\ w)$
by (*metis Initprop(2) StateUntilEqvWPrevChop UntilRule inteq-reflection*)

lemma *PiChopDist1*:
$\vdash ((init\ w)\ \Pi\ (f;g) \vee ((init\ w)\ \Pi\ (f \wedge finite) \wedge inf)) =$
$(((init\ w)\ \Pi\ f);((init\ w) \wedge\ (init\ w)\ \Pi\ g)\ )$
proof −
have *1*: $\vdash f;g = (f\frown g \vee (f \wedge inf))$
by (*simp add*: *ChopSChopdef*)
have *2*: $\vdash (init\ w)\ \Pi\ (f;g) = ((init\ w)\ \Pi\ (f\frown g) \vee (init\ w)\ \Pi\ (f \wedge inf))$
by (*metis 1 PiOr int-eq*)
have *3*: $\vdash (init\ w)\ \Pi\ (f\frown g) = ((init\ w)\ \Pi\ f)\frown((init\ w) \wedge (init\ w)\ \Pi\ g)$

by (*simp add*: *PiSChopDist*)
have *4*: $\vdash ((init\ w)\ \Pi\ f);((init\ w) \wedge\ (init\ w)\ \Pi\ g) =$
$(((init\ w)\ \Pi\ f)\frown((init\ w) \wedge (init\ w)\ \Pi\ g) \vee ((init\ w)\ \Pi\ f \wedge inf))$
by (*simp add*: *ChopSChopdef*)
have *41*: $\vdash f = (f \wedge (finite \vee inf))$
unfolding *finite-d-def* **by** *fastforce*
have *5*: $\vdash (init\ w)\ \Pi\ f =\ (init\ w)\ \Pi\ (f \wedge (finite \vee inf))$
by (*simp add*: *41 PiEqvRule*)
have *6*: $\vdash ((init\ w)\ \Pi\ f \wedge inf) =$
$(((init\ w)\ \Pi\ (f \wedge finite) \wedge inf) \vee ((init\ w)\ \Pi\ (f \wedge inf) \wedge inf))$
by (*metis AndInfEqvChopFalse OrChopEqvRule OrFiniteInf PiOr inteq-reflection*)
have *7*: $\vdash ((init\ w)\ \Pi\ (f \wedge inf) \wedge inf) = ((init\ w)\ \Pi\ (f \wedge inf))$
by (*metis EmptySChop PiImpRule PiInfImpInf Prop01 Prop04 Prop05 Prop10 int-iffD1 lift-imp-trans*)
show *?thesis*
using *2 3 4 6 7* **by** *fastforce*
qed

lemma *PiChopDist2*:
$\vdash ((init\ w)\ \Pi\ (f;g)) =$
$(((init\ w)\ \Pi\ (f \wedge finite)\ )\frown((init\ w) \wedge (init\ w)\ \Pi\ g) \vee ((init\ w)\ \Pi\ (f \wedge inf) \wedge inf))$
proof −
have *1*: $\vdash (init\ w)\ \Pi\ (f;g) = ((init\ w)\ \Pi\ (f\frown g) \vee (init\ w)\ \Pi\ (f \wedge inf))$
by (*metis ChopSChopdef PiOr inteq-reflection*)
have *2*: $\vdash (init\ w)\ \Pi\ (f\frown g) = ((init\ w)\ \Pi\ f)\frown((init\ w) \wedge (init\ w)\ \Pi\ g)$
by (*simp add*: *PiSChopDist*)
have *3*: $\vdash ((init\ w)\ \Pi\ f)\frown((init\ w) \wedge (init\ w)\ \Pi\ g) =$
$((init\ w)\ \Pi\ f \wedge finite);((init\ w) \wedge (init\ w)\ \Pi\ g)$
by (*simp add*: *schop-d-def*)
have *31*: $\vdash f = (\ (f \wedge finite) \vee (f \wedge inf))$
unfolding *finite-d-def* **by** *fastforce*
have *32*: $\vdash (init\ w)\ \Pi\ f = (init\ w)\ \Pi\ (\ (f \wedge finite) \vee (f \wedge inf))$
using *31 PiEqvRule* **by** *blast*
have *33*: $\vdash (init\ w)\ \Pi\ (\ (f \wedge finite) \vee (f \wedge inf)) = ((init\ w)\ \Pi\ (f \wedge finite) \vee (init\ w)\ \Pi\ (f \wedge inf))$
by (*simp add*: *PiOr*)
have *34*: $\vdash \neg((init\ w)\ \Pi\ (f \wedge inf) \wedge finite)$
unfolding *finite-d-def* **using** *PiAnd*[*of LIFT init w f LIFT inf*] *PiInfImpInf*[*of LIFT init w*]
by *fastforce*
have *4*: $\vdash ((init\ w)\ \Pi\ f \wedge finite) = ((init\ w)\ \Pi\ (f \wedge finite) \wedge finite)$
using *32 33 34* **by** *fastforce*
have *5*: $\vdash (init\ w)\ \Pi\ (f \wedge inf) = ((init\ w)\ \Pi\ (f \wedge inf) \wedge inf)$
by (*metis PiAnd PiInfImpInf Prop10 Prop12 int-eq int-iffD2*)
show *?thesis*
by (*metis 1 2 4 5 int-eq schop-d-def*)
qed

lemma *InfEqvNotForallNotLen*:
$\vdash inf = (\forall n.\ \neg\ (len\ n))$
proof −
have *1*: $\vdash finite = (\exists n.\ len\ n)$

by (*simp add*: *Finite-exist-len*)
have *2*: $\vdash (\neg\ finite) = (\neg(\exists\, n.\ len\ n))$
using *1*
by (*metis NotEqvYieldsMore int-eq*)
have *3*: $\vdash (\neg(\exists\, n.\ len\ n)) = (\forall\, n.\ \neg\ (len\ n))$
by *fastforce*
show *?thesis* **using** *2 3* **by** (*metis InfEqvNotFinite int-eq*)
qed

lemma *PiEx*:
$\vdash f\ \Pi\ (\exists\, n.\ g\ n) = (\exists\, n.\ f\ \Pi\ (g\ n))$
unfolding *Valid-def* **by** (*auto simp add*: *pi-d-def*)

lemma *PiAll*:
$\vdash f\ \Pi\ (\forall\, n.\ g\ n) = (\forall\, n.\ f\ \Pi\ (g\ n))$
unfolding *Valid-def* **by** (*auto simp add*: *pi-d-def*)

lemma *PiLenSuc*:
$\vdash (init\ w)\ \Pi\ (len\ (Suc\ n)) = ((init\ w)\ \Pi\ skip)^\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ (len\ n)\ )$
proof −
have *1*: $\vdash len\ (Suc\ n) = skip^\frown(len\ n)$
by (*metis NextSChopdef len-d-def next-d-def wpow-Suc*)
have *2*: $\vdash (init\ w)\ \Pi\ (skip^\frown(len\ n)) = ((init\ w)\ \Pi\ skip)^\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ (len\ n)\ )$
by (*simp add*: *PiSChopDist*)
show *?thesis* **by** (*metis 1 2 int-eq*)
qed

lemma *PiImpDiamond*:
$\vdash f\ \Pi\ g \longrightarrow \Diamond\ f$
by (*meson PiEqvDiamondUPi Prop12 int-iffD1*)

lemma *PiMPA*:
assumes $\vdash f\ \Pi\ g1$
$\vdash f\ \Pi\ (g1 \longrightarrow g2)$
shows $\vdash f\ \Pi\ g2$
using *assms*
by (*meson MP PiDc Prop09 UpiAndImp*)

lemma *PiMPB*:
assumes $\vdash f\ \Pi\ g1$
$\vdash g1 \longrightarrow g2$
shows $\vdash f\ \Pi\ g2$
using *assms*

by (*metis MP PiAnd Prop10 Prop12 int-iffD2 inteq-reflection*)

lemma *PiMPBC*:
assumes $\vdash$ *f1* $\longrightarrow$ *f2* Π *g1*
$\vdash$ *g1* $\longrightarrow$ *g2*
shows $\vdash$ *f1* $\longrightarrow$ *f2* Π *g2*
using *assms*
by (*meson PiImpRule lift-imp-trans*)

lemma *SlideInitSFin*:
$\vdash f\frown(\ (init\ w) \wedge g) = (f \wedge sfin\ w)\frown g$
proof −
have *1*: $\vdash$ *sfin* (*init w*) = *sfin* (*w*)
by (*metis InitAndEmptyEqvAndEmpty SFinEqvTrueSChopAndEmpty int-eq*)
show *?thesis*
by (*metis 1 AndSFinSChopEqvStateAndSChop inteq-reflection*)
qed

lemma *UPiSChop*:
$\vdash$ (*init w*) Π^u ($f\frown g$) = ($\Box$(*init* ($\neg w$)) $\vee$ ((*init w*) Π f $\wedge$ *sfin w*)$\frown$((*init w*) Π g))
proof −
have *1*: $\vdash$ (*init w*) Π^u ($f\frown g$) = ($\Box$ (*init* ($\neg w$)) $\vee$ (*init w*) Π ($f\frown g$))
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi int-eq*)
have *2*: $\vdash$ (*init w*) Π ($f\frown g$) = ((*init w*) Π f)$\frown$((*init w*) $\wedge$ (*init w*) Π g)
by (*simp add*: *PiSChopDist*)
have *3*: $\vdash$ ((*init w*) Π f)$\frown$((*init w*) $\wedge$ (*init w*) Π g) =
((*init w*) Π f $\wedge$ *sfin w*)$\frown$((*init w*) Π g)
by (*simp add*: *SlideInitSFin*)
show *?thesis*
using *1 2 3* **by** (*meson Prop06*)
qed

lemma *PiUPiSChop*:
$\vdash$ ((*init w*) Π f $\wedge$ *sfin w*)$\frown$((*init w*) Π g) = ((*init w*) Π^u f $\wedge$ *sfin w*)$\frown$((*init w*) Π^u g)
proof −
have *1*: $\vdash$ ((*init w*) Π f) = ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u f)
by (*simp add*: *PiEqvDiamondUPi*)
have *2*: $\vdash$ ((*init w*) Π f $\wedge$ *sfin w*) = ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u f $\wedge$ *sfin w*)
using *1* **by** *auto*
have *3*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ *sfin w*) = *sfin w*
by (*metis InitAndEmptyEqvAndEmpty Prop10 SChopAndA SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond*
inteq-reflection lift-and-com)
have *4*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u f $\wedge$ *sfin w*) = ((*init w*) Π^u f $\wedge$ *sfin w*)
using *3* **by** *auto*
have *5*: $\vdash$ ((*init w*) Π g) = ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u g)
by (*simp add*: *PiEqvDiamondUPi*)

have *6*: $\vdash$ (*init w* $\wedge$ (*init w*) Π *g*) = (*init w* $\wedge$ $\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *g*)
using *5* **by** *auto*
have *7*: $\vdash$ (*init w* $\wedge$ $\Diamond$ (*init w*)) = *init w*
by (*meson NowImpDiamond Prop10 Prop11*)
have *8*: $\vdash$ (*init w* $\wedge$ $\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *g*) = (*init w* $\wedge$ (*init w*) Π^u *g*)
using *7* **by** *auto*
show *?thesis*
by (*metis 2 4 6 8 SlideInitSFin inteq-reflection*)
qed

lemma *PiFiniteAbsorb*:
$\vdash$ (*g* Π (*f* $\wedge$ *finite*) $\wedge$ *finite*) = (*g* Π *f* $\wedge$ *finite*)
proof −
have *1*: $\vdash$ (*g* Π (*f* $\wedge$ *finite*) $\wedge$ *finite*) $\longrightarrow$ (*g* Π *f* $\wedge$ *finite*)
by (*metis FiniteImp PiAnd PiAndPiImpPiAnd Prop01 Prop05 Prop12 inteq-reflection lift-and-com*)
have *31*: $\vdash$ *f* = ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*))
unfolding *finite-d-def* **by** *fastforce*
have *32*: $\vdash$ *g* Π *f* = *g* Π ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*))
using *31 PiEqvRule* **by** *blast*
have *33*: $\vdash$ *g* Π ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*)) = (*g* Π (*f* $\wedge$ *finite*) $\vee$ *g* Π (*f* $\wedge$ *inf*))
by (*simp add*: *PiOr*)
have *34*: $\vdash$ ¬(*g* Π (*f* $\wedge$ *inf*) $\wedge$ *finite*)
unfolding *finite-d-def* **using** *PiAnd*[*of g f LIFT inf*] *PiInfImpInf*[*of g*]
by *fastforce*
have *4*: $\vdash$ (*g* Π *f* $\wedge$ *finite*) = (*g* Π (*f* $\wedge$ *finite*) $\wedge$ *finite*)
using *32 33 34* **by** *fastforce*
have *2*: $\vdash$ (*g* Π *f* $\wedge$ *finite*) $\longrightarrow$ (*g* Π (*f* $\wedge$ *finite*) $\wedge$ *finite*)
by (*simp add*: *4 int-iffD1*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *PiInfAbsorb*:
$\vdash$ (*g* Π (*f* $\wedge$ *inf*) $\wedge$ *inf*) = (*g* Π (*f* $\wedge$ *inf*))
proof −
have *1*: $\vdash$ (*g* Π (*f* $\wedge$ *inf*) $\wedge$ *inf*) $\longrightarrow$ (*g* Π (*f* $\wedge$ *inf*))
by *fastforce*
have *2*: $\vdash$ (*g* Π (*f* $\wedge$ *inf*)) $\longrightarrow$ (*g* Π (*f* $\wedge$ *inf*) $\wedge$ *inf*)
by (*metis PiAnd PiInfImpInf Prop10 Prop12 int-iffD1 lift-imp-trans*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *PiMoreImpMore*:
$\vdash$ (*g* Π *more*) $\longrightarrow$ *more*
unfolding *Valid-def pi-d-def itl-defs pifilt-def sfxfilt-def*
by *auto*

(*metis enat-min-eq-0-iff length-nfilter-le min-def ndropns-nlength*)

lemma *PiMoreAbsorb*:
$\vdash (g\ \Pi\ (f \wedge more\ )) \wedge more) = (g\ \Pi\ (f \wedge more))$
proof −
have *1*: $\vdash (g\ \Pi\ (f \wedge more\ )) \wedge more) \longrightarrow (g\ \Pi\ (f \wedge more))$
by *auto*
have *2*: $\vdash (g\ \Pi\ (f \wedge more))\ ) \longrightarrow (g\ \Pi\ (f \wedge more\ )) \wedge more)$
by (*metis PiMoreImpMore PiAnd Prop10 Prop12 int-iffD1 inteq-reflection*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *EmptyImpPiEmpty*:
$\vdash (g\ \Pi\ f \wedge empty\ ) \longrightarrow (g\ \Pi\ empty)$
unfolding *Valid-def pi-d-def itl-defs pifilt-def sfxfilt-def*
by *auto*
(*metis ile0-eq length-nfilter-le ndropns-nlength*)

lemma *PiEmptyAbsorb*:
$\vdash (g\ \Pi\ (f \wedge empty) \wedge empty) = (g\ \Pi\ f \wedge empty\ )$
proof −
have *1*: $\vdash\ (g\ \Pi\ (f \wedge empty) \wedge empty) \longrightarrow (g\ \Pi\ f \wedge empty\ )$
using *PiAnd* **by** *fastforce*
have *2*: $\vdash (g\ \Pi\ f \wedge empty\ ) \longrightarrow (g\ \Pi\ (f \wedge empty) \wedge empty)$
using *EmptyImpPiEmpty PiAnd* **by** *fastforce*
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *SlideInitSFinVar1*:
$\vdash (init\ w)\ \Pi\ (((F \wedge more) \wedge finite);X) =$
$(((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);((init\ w)\ \Pi\ X)$
proof −
have *1*: $\vdash (init\ w)\ \Pi\ (((F \wedge more) \wedge finite);X) =$
$((init\ w)\ \Pi\ (F \wedge more) \wedge finite);((init\ w) \wedge (init\ w)\ \Pi\ X)$
by (*metis PiSChopDist schop-d-def*)
have *2*: $\vdash ((init\ w)\ \Pi\ (F \wedge more) \wedge finite);((init\ w) \wedge (init\ w)\ \Pi\ X) =$
$(((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);(\ (init\ w)\ \Pi\ X)$
using *SlideInitSFin*[*of LIFT* $(init\ w)\ \Pi\ (F \wedge more)\ w$ *LIFT* $(init\ w)\ \Pi\ X$] **unfolding** *schop-d-def*
by *blast*
show *?thesis* **using** *1 2* **by** (*metis int-eq*)
qed

lemma *UPiAbsorp*:
$\vdash (((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);((init\ w)\ \Pi^u\ X) =$

$(((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);((init\ w)\ \Pi\ X)$
proof −
have *0*: $\vdash ((init\ w)\ \Pi^u\ X) = (\Box\ (init\ (\neg w)) \vee\ (init\ w)\ \Pi\ X)$
by (*metis Initprop(2) UPiEqvBoxOrPi inteq-reflection*)
have *1*: $\vdash (((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);((init\ w)\ \Pi^u\ X) =$
$(((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);(\Box\ (init\ (\neg w)) \vee\ (init\ w)\ \Pi\ X)$
using *0 RightChopEqvChop* **by** *blast*
have *2*: $\vdash (((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);(\Box\ (init\ (\neg w)) \vee\ (init\ w)\ \Pi\ X) =$
$(((init\ w)\ \Pi\ (F \wedge more)) \wedge finite);((init\ w) \wedge (\Box\ (init\ (\neg w)) \vee\ (init\ w)\ \Pi\ X))$
using *SlideInitSFin*[*of LIFT* $(init\ w)\ \Pi\ (F \wedge more)$ *w LIFT* $(\Box\ (init\ (\neg w)) \vee\ (init\ w)\ \Pi\ X)$]
unfolding *schop-d-def*
by (*metis inteq-reflection*)
have *3*: $\vdash ((init\ w) \wedge (\Box\ (init\ (\neg w)) \vee\ (init\ w)\ \Pi\ X)) =$
$((init\ w) \wedge (\ (init\ w)\ \Pi\ X))$
using *BoxElim Initprop(2)* **by** *fastforce*
have *4*: $\vdash (((init\ w)\ \Pi\ (F \wedge more)) \wedge finite);((init\ w) \wedge (\Box\ (init\ (\neg w)) \vee\ (init\ w)\ \Pi\ X)) =$
$(((init\ w)\ \Pi\ (F \wedge more)) \wedge finite);((init\ w) \wedge\ \ (init\ w)\ \Pi\ X)$
using *3 RightChopEqvChop* **by** *blast*
have *5*: $\vdash (((init\ w)\ \Pi\ (F \wedge more)) \wedge finite);((init\ w) \wedge\ \ (init\ w)\ \Pi\ X) =$
$(((init\ w)\ \Pi\ (F \wedge more) \wedge sfin\ w) \wedge finite);((init\ w)\ \Pi\ X)$
using *SlideInitSFin*[*of LIFT* $((init\ w)\ \Pi\ (F \wedge more))$ *w LIFT* $(init\ w)\ \Pi\ X$]
unfolding *schop-d-def*
by *auto*
show *?thesis*
by (*metis 1 2 4 5 inteq-reflection*)
qed

lemma *PiStateFinite*:
$\vdash (init\ w)\ \Pi\ finite = \Diamond\ ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg w)))))$
proof −
have *1*: $\vdash\ \ (init\ w)\ \Pi\ finite = (init\ w)\ \Pi\ (\#True \frown empty)$
by (*metis DiamondEmptyEqvFinite DiamondSChopdef PiEqvRule int-eq*)
have *2*: $\vdash\ \ (init\ w)\ \Pi\ (\#True \frown empty) = ((init\ w)\ \Pi\ \#True) \frown ((init\ w) \wedge (init\ w)\ \Pi\ empty)$
by (*simp add*: *PiSChopDist*)
have *3*: $\vdash ((init\ w) \wedge (init\ w)\ \Pi\ empty) = ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg w))))$
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection*)
have *4*: $\vdash (init\ w)\ \Pi\ \#True = \Diamond\ (init\ w)$
by (*simp add*: *PiTrueEqvDiamond*)
have *5*: $\vdash ((init\ w)\ \Pi\ \#True) \frown ((init\ w) \wedge (init\ w)\ \Pi\ empty) =$
$(\ \Diamond\ (init\ w)) \frown\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg w))))$
by (*simp add*: *3 4 SChopEqvSChop*)
have *6*: $\vdash (\ \Diamond\ (init\ w)) \frown\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg w)))) =$
$(\ \Diamond\ (init\ w) \wedge sfin\ w) \frown (wnext\ (\Box\ (init\ (\neg w))))$
by (*simp add*: *SlideInitSFin*)
have *7*: $\vdash (\ \Diamond\ (init\ w) \wedge sfin\ w) = sfin\ w$
by (*metis DiamondSChopdef InitAndEmptyEqvAndEmpty Prop10 SChopAndA SFinEqvTrueSChopAndEmpty*
inteq-reflection lift-and-com)

have *8*: $\vdash$ ($\diamond$ (*init w*) $\wedge$ *sfin w*)$\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*)))) =
(*sfin w*) $\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*))))
using *7 LeftSChopEqvSChop* **by** *blast*
have *9*: $\vdash$ (*sfin w*) $\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*)))) = #*True* $\frown$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))
by (*metis SlideInitSFin int-eq int-simps*(*17*))
show *?thesis*
by (*metis 1 2 3 4 6 7 9 TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *PiStateInf*:
$\vdash$ (*init w*) Π *inf* = ($\diamond$ (*init w*) $\wedge$ $\Box$ ((*init w*) $\longrightarrow$ $\bigcirc$ ($\diamond$ (*init w*))))
proof −
have *1*: $\vdash$ (*init w*) Π *inf* = (*init w*) Π ($\neg$ *finite*)
by (*simp add*: *InfEqvNotFinite PiEqvRule*)
have *2*: $\vdash$ (*init w*) Π ($\neg$ *finite*) = ($\neg$ ((*init w*) Π^u *finite*))
by (*simp add*: *NotPiEqvNotUPi*)
have *3*: $\vdash$ ((*init w*) Π^u *finite*) = ($\Box$ (*init* ($\neg$ *w*)) $\vee$ (*init w*) Π *finite*)
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi inteq-reflection*)
have *4*: $\vdash$ ($\Box$ (*init* ($\neg$ *w*)) $\vee$ (*init w*) Π *finite*) =
($\Box$ (*init* ($\neg$ *w*)) $\vee$ $\diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*))))))
using *PiStateFinite* **by** *fastforce*
have *5*: $\vdash$ ($\neg$ ((*init w*) Π^u *finite*)) =
($\neg$ ($\Box$ (*init* ($\neg$ *w*)) $\vee$ $\diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))))
using *3 4* **by** *fastforce*
have *6*: $\vdash$ ($\neg$ ($\Box$ (*init* ($\neg$ *w*)) $\vee$ $\diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))) =
($\neg$($\Box$ (*init* ($\neg$ *w*))) $\wedge$ $\neg$($\diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*))))))
by *force*
have *7*: $\vdash$ ($\neg$($\Box$ (*init* ($\neg$ *w*)))) = $\diamond$ (*init w*)
unfolding *always-d-def*
by (*metis Initprop*(*2*) *int-simps*(*4*) *inteq-reflection*)
have *8*: $\vdash$ ($\neg$($\diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))) =
$\Box$($\neg$ ((*init* (*w*)) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))
by (*simp add*: *always-d-def*)
have *9*: $\vdash$ ($\neg$ ((*init* (*w*)) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))) =
($\neg$(*init* (*w*)) $\vee$ $\neg$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))
by *force*
have *10*: $\vdash$ ($\neg$(*init* (*w*))) = (*init* ($\neg$ *w*))
by (*simp add*: *Initprop*(*2*))
have *11*: $\vdash$ ($\neg$ (*wnext* ($\Box$ (*init* ($\neg$*w*))))) = $\bigcirc$($\neg$ (($\Box$ (*init* ($\neg$*w*)))))
by (*simp add*: *wnext-d-def*)
have *12*: $\vdash$ $\bigcirc$($\neg$ (($\Box$ (*init* ($\neg$*w*))))) = $\bigcirc$ ($\diamond$ (*init w*))
by (*simp add*: *7 NextEqvNext*)
have *13*: $\vdash$ ($\neg$(*init* (*w*)) $\vee$ $\neg$ (*wnext* ($\Box$ (*init* ($\neg$*w*))))) = ((*init* *w*) $\longrightarrow$ $\bigcirc$ ($\diamond$ (*init w*)))
using *10 11 12* **by** *fastforce*
have *14*: $\vdash$ $\Box$($\neg$ ((*init* (*w*)) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))) =
$\Box$ ((*init w*) $\longrightarrow$ $\bigcirc$ ($\diamond$ (*init w*)))
by (*metis 13 8 9 inteq-reflection*)
have *15*: $\vdash$ ($\neg$($\Box$ (*init* ($\neg$ *w*))) $\wedge$ $\neg$($\diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*))))))) =
($\diamond$ (*init w*) $\wedge$ $\Box$ ((*init* *w*) $\longrightarrow$ $\bigcirc$ ($\diamond$ (*init w*))))

by (*metis 13 6 7 8 9 inteq-reflection*)
show *?thesis*
by (*metis 1 15 2 5 6 int-eq*)
qed

lemma *UPiPiState*:
$\vdash$ (*init w*) Π^u ((*init w*) Π *f*) = (*init w*) Π^u *f*
proof −
have *1*: $\vdash$ (*init w*) Π^u ((*init w*) Π *f*) = ($\Box$(*init* ($\neg$*w*)) $\vee$ (*init w*) Π ((*init w*) Π *f*))
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi int-eq*)
have *2*: $\vdash$ (*init w*) Π ((*init w*) Π *f*) = ((*init w*) Π *f*)
by (*metis PiAssoc Prop10 StateEqvBi StateImpBi inteq-reflection*)
have *3*: $\vdash$ ($\Box$(*init* ($\neg$*w*)) $\vee$ (*init w*) Π ((*init w*) Π *f*)) =
($\Box$(*init* ($\neg$*w*)) $\vee$ ((*init w*) Π *f*))
using *2* **by** *auto*
have *4*: $\vdash$ ($\Box$(*init* ($\neg$*w*)) $\vee$ ((*init w*) Π *f*)) = (*init w*) Π^u *f*
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi int-eq*)
show *?thesis*
by (*metis 1 3 4 int-eq*)
qed

lemma *PiUPiState*:
$\vdash$ (*init w*) Π ((*init w*) Π^u *f*) = (*init w*) Π *f*
proof −
have *1*: $\vdash$ (*init w*) Π ((*init w*) Π^u *f*) = ($\Diamond$(*init w*) $\wedge$ (*init w*) Π^u ((*init w*) Π^u *f*))
by (*simp add*: *PiEqvDiamondUPi*)
have *2*: $\vdash$ (*init w*) Π^u ((*init w*) Π^u *f*) = ((*init w*) Π^u *f*)
by (*metis* (*no-types*, *lifting*) *NotUPiEqvNotPi UPiPiState inteq-reflection upi-d-def*)
have *3*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u ((*init w*) Π^u *f*)) =
($\Diamond$(*init w*) $\wedge$ ((*init w*) Π^u *f*))
using *2* **by** *auto*
have *4*: $\vdash$ ($\Diamond$(*init w*) $\wedge$ ((*init w*) Π^u *f*)) = (*init w*) Π *f*
by (*meson PiEqvDiamondUPi Prop11*)
show *?thesis*
by (*metis 1 3 4 int-eq*)
qed

lemma *PiStateFiniteAndFinite*:
$\vdash$ ((*init w*) Π *finite* $\wedge$ *finite*) = ($\Diamond$ (*init w*) $\wedge$ *finite*)
proof −
have *1*: $\vdash$ ((*init w*) Π *finite* $\wedge$ *finite*) $\longrightarrow$ ($\Diamond$ (*init w*) $\wedge$ *finite*)
using *PiImpDiamond* **by** *fastforce*
have *2*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ *finite*) $\longrightarrow$ ((*init w*) Π *finite* $\wedge$ *finite*)
by (*metis PiFiniteAbsorb PiTrueEqvDiamond TrueW int-simps*(*13*) *int-simps*(*17*) *inteq-reflection*)
show *?thesis*
using *1 2 int-iffI* **by** *blast*

qed

lemma *PiStateState*:
$\vdash$ (*init w*) Π (*init w*) = (*init w*) Π #*True*
by (*metis BoxElim PiImpDiamond PiMPBC PiTrueEqvDiamond Prop11 StatePiBoxState int-simps*(*17*) *inteq-reflection*)

lemma *StateAndPiState*:
$\vdash$ ((*init w*) $\wedge$ (*init w*) Π (*empty* $\wedge$ *w*)) = ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$*w*))))
proof −
have *2*: $\vdash$ ((*init w*) $\wedge$ *init w* Π *empty*) = (*init w* $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*))))
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection*)
have *5*: $\vdash$ (*init w*) Π #*True* = $\Diamond$ (*init w*)
by (*simp add*: *PiTrueEqvDiamond*)
show *?thesis*
by (*metis 2 5 InitAndEmptyEqvAndEmpty PiAnd PiImpDiamond PiStateState Prop10 inteq-reflection lift-and-com*)
qed

lemma *PiStateFiniteSfin*:
$\vdash$ ((*init w*) Π *finite* $\wedge$ *sfin w*) = *sfin w*
proof −
have *1*: $\vdash$ ((*init w*) Π *finite* $\wedge$ *sfin w*) =
(((*init w*) Π *finite* $\wedge$ *finite*) $\wedge$ *sfin w*)
by (*metis AndSFinEqvSChopAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have *2*: $\vdash$ (((*init w*) Π *finite* $\wedge$ *finite*) $\wedge$ *sfin w*) = (($\Diamond$ (*init w*) $\wedge$ *finite*) $\wedge$ *sfin w*)
using *PiStateFiniteAndFinite* **by** *fastforce*
have *3*: $\vdash$ (($\Diamond$ (*init w*) $\wedge$ *finite*) $\wedge$ *sfin w*) = (($\Diamond$ (*init w*)) $\wedge$ *sfin w*)
by (*metis AndSFinEqvSChopAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have *4*: $\vdash$ (($\Diamond$ (*init w*)) $\wedge$ *sfin w*) = *sfin w*
by (*metis 3 InitAndEmptyEqvAndEmpty Prop11 Prop12 SChopAndA SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection*)
show *?thesis*
by (*metis 1 2 3 4 inteq-reflection*)
qed

lemma *PiStateSFin*:
$\vdash$ (*init w*) Π (*sfin w*) = (*sfin w*);(*wnext* ($\Box$ (*init* ($\neg$*w*))))
proof −
have *1*: $\vdash$ (*sfin w*) = *finite*;(*empty* $\wedge$ *w*)
by (*metis ChopAndCommute SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection sometimes-d-def*)
have *2*: $\vdash$ (*init w*) Π (*finite*;(*empty* $\wedge$ *w*)) =

((*init w*) Π *finite* ∧ *finite*);((*init w*) ∧ (*init w*) Π (*empty* ∧ *w*))
using *PiSChopDist*[*of w LIFT finite LIFT* (*empty* ∧ *w*)] **unfolding** *schop-d-def*
by *auto*
have *3*: ⊢ ((*init w*) Π *finite* ∧ *finite*);((*init w*) ∧ (*init w*) Π (*empty* ∧ *w*)) =
((*init w*) Π *finite* ∧ *finite*);((*init w*) ∧ *wnext* (□ (*init* (¬*w*))))
using *RightChopEqvChop StateAndPiState* **by** *blast*
have *4*: ⊢ (*sfin w* ∧ *finite*) = *sfin w*
by (*metis 1 ChopAndA DiamondEmptyEqvFinite Prop10 inteq-reflection sometimes-d-def*)
have *5*: ⊢ ((*init w*) Π *finite* ∧ *finite*);((*init w*) ∧ *wnext* (□ (*init* (¬*w*)))) =
((*init w*) Π *finite* ∧ *sfin w*);(*wnext* (□ (*init* (¬*w*))))
using *SlideInitSFin*[*of LIFT* (*init w*) Π *finite w LIFT wnext* (□ (*init* (¬*w*)))] *4*
unfolding *schop-d-def*
by (*metis Prop10 Prop12 int-eq int-iffD2 lift-and-com*)
have *6*: ⊢ ((*init w*) Π *finite* ∧ *sfin w*);(*wnext* (□ (*init* (¬*w*)))) = (*sfin w*);(*wnext* (□ (*init* (¬*w*))))
using *LeftChopEqvChop PiStateFiniteSfin* **by** *blast*
show *?thesis*
by (*metis 1 2 3 5 6 int-eq*)
qed

3.9.5 SChopstar and Pi

lemma *PiChopstarhelp2a*:
⊢ (*w* ∧ *empty*)⌢(*fpower* ((*f*⌢(*w* ∧ *empty*)) ∧ *more*) *k*) =
(*fpower* (((*w* ∧ *empty*)⌢*f*) ∧ *more*) *k*) ⌢(*w* ∧ *empty*)
proof (*induction k*)
case *0*
then show *?case*
proof −
have *1*: ⊢ (*w* ∧ *empty*) ⌢ *empty* = *empty* ⌢ (*w* ∧ *empty*)
by (*metis EmptySChop InitAndEmptyEqvAndEmpty StateAndEmptySChop int-eq*)
show *?thesis*
by (*metis 1 fpower-d-def wpow-0*)
qed
next
case (*Suc k*)
then show *?case*
proof −
have *1*: ⊢ (*w* ∧ *empty*)⌢*fpower* (*f*⌢(*w* ∧ *empty*) ∧ *more*) (*Suc k*) =
(*w* ∧ *empty*)⌢((*f*⌢(*w* ∧ *empty*) ∧ *more*)⌢ *fpower* (*f*⌢(*w* ∧ *empty*) ∧ *more*) *k*)
unfolding *fpower-d-def*
by (*simp add*: *schop-d-def*)
have *11*: ⊢ (*f* ∧ *more*)⌢(*w* ∧ *empty*) = (*sfin w* ∧ (*f* ∧ *more*))
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty int-eq*)
have *12*: ⊢ (*f*⌢(*w* ∧ *empty*)) = (*sfin w* ∧ *f*)
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection*)
have *13*: ⊢ (*f*⌢(*w* ∧ *empty*) ∧ *more*) = ((*sfin w* ∧ *f*) ∧ *more*)
using *12* **by** *auto*
have *14*: ⊢ ((*sfin w* ∧ *f*) ∧ *more*) = (*sfin w* ∧ (*f* ∧ *more*))
by *fastforce*
have *2*: ⊢ (*f*⌢(*w* ∧ *empty*) ∧ *more*) = (*f* ∧ *more*)⌢(*w* ∧ *empty*)

using *11 13* **by** *fastforce*
have *20*: $\vdash((f\frown(w \wedge empty) \wedge more)\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k) =$
$((f \wedge more)\frown(w \wedge empty))\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k$
by (*simp add*: *2 LeftSChopEqvSChop*)
have *21*: $\vdash ((f \wedge more)\frown(w \wedge empty))\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k =$
$(f \wedge more)\frown((w \wedge empty)\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k)$
by (*meson Prop11 SChopAssoc*)
have *22*: $\vdash (f \wedge more)\frown((w \wedge empty)\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k) =$
$((f \wedge more)\frown (fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty)))$
by (*simp add*: *RightSChopEqvSChop Suc.IH*)
have *23*: $\vdash (w \wedge empty)\frown((f\frown(w \wedge empty) \wedge more)\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k) =$
$(w \wedge empty)\frown(((f \wedge more) \frown (w \wedge empty))\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k)$
using *20 RightSChopEqvSChop* **by** *blast*
have *24*: $\vdash (w \wedge empty)\frown(((f \wedge more) \frown (w \wedge empty))\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k) =$
$(w \wedge empty)\frown((f \wedge more) \frown ((w \wedge empty)\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k))$
using *21 RightSChopEqvSChop* **by** *blast*
have *25*: $\vdash (w \wedge empty)\frown((f \wedge more) \frown ((w \wedge empty)\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k)) =$
$(w \wedge empty)\frown((f \wedge more)\frown (fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty)))$
using *23 22 RightSChopEqvSChop* **by** *blast*
have *3*: $\vdash (w \wedge empty)\frown((f\frown(w \wedge empty) \wedge more)\frown fpower\ (f\frown(w \wedge empty) \wedge more)\ k) =$
$(w \wedge empty)\frown((f \wedge more)\frown (fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty)))$
using *23 24 25* **by** *fastforce*
have *4*: $\vdash (w \wedge empty)\frown((f \wedge more)\frown (fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty))) =$
$((w \wedge empty)\frown(f \wedge more))\frown ((fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty)))$
using *SChopAssoc* **by** *blast*
have *6*: $\vdash ((w \wedge empty)\frown(f \wedge more))\frown ((fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty))) =$
$((w \wedge empty)\frown f \wedge more)\frown ((fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty)))$
by (*meson EmptyAndSChopAndMoreEqvAndSChop LeftSChopEqvSChop*)
have *7*: $\vdash ((w \wedge empty)\frown f \wedge more)\frown ((fpower\ ((w \wedge empty)\frown f \wedge more)\ k\frown(w \wedge empty))) =$
$((fpower\ ((w \wedge empty)\frown f \wedge more)\ (Suc\ k)\frown(w \wedge empty)))$
by (*metis SChopAssoc fpower-d-def schop-d-def wpow-Suc*)
show *?thesis* **using** *1 3 4 6 7*
by (*metis inteq-reflection*)
qed
qed

lemma *PiChopstarhelp2*:
$\vdash (w \wedge empty)\frown(schopstar(f\frown(w \wedge empty))) = (schopstar((w \wedge empty)\frown f))\frown(w \wedge empty)$
proof −
have *1*: $\vdash (w \wedge empty)\frown(schopstar(f\frown(w \wedge empty))) =$
$(w \wedge empty)\frown(\exists\, k.\ fpower\ (\ (f\frown(w \wedge empty)) \wedge more)\ k)$
by (*simp add*: *schopstar-d-def fpowerstar-d-def*)
have *2*: $\vdash (w \wedge empty)\frown(\exists\, k.\ fpower\ (\ (f\frown(w \wedge empty)) \wedge more)\ k) =$
$(\exists\, k.\ (w \wedge empty)\frown(fpower\ (\ (f\frown(w \wedge empty)) \wedge more)\ k))$
using *SChopExist* **by** *fastforce*
have *3*: $\vdash (\exists\, k.\ (w \wedge empty)\frown(fpower\ (\ (f\frown(w \wedge empty)) \wedge more)\ k)) =$
$(\exists\, k.\ (fpower\ ((w \wedge empty)\frown f \wedge more)\ k)\frown(w \wedge empty))$
by (*simp add*: *ExEqvRule PiChopstarhelp2a*)
have *4*: $\vdash (\exists\, k.\ (fpower\ ((w \wedge empty)\frown f \wedge more)\ k)\frown(w \wedge empty)) =$
$(\exists\, k.\ (fpower\ ((w \wedge empty)\frown f \wedge more)\ k))\frown(w \wedge empty)$

using *ExistSChop* **by** *fastforce*
have *5*: $\vdash (\exists k.\ (fpower\ ((w \wedge empty)^\frown f \wedge more)\ k))^\frown(w \wedge empty) =$
$(schopstar((w \wedge empty)^\frown f))^\frown(w \wedge empty)$
by (*simp add*: *schopstar-d-def fpowerstar-d-def*)
show *?thesis*
by (*metis 1 2 3 4 5 int-eq*)
qed

lemma *PiFPowerSuca*:
$\vdash (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f)^\frown((init\ w) \wedge (init\ w)\ \Pi\ (fpower\ f\ k))$
unfolding *fpower-d-def* **by** (*metis PiSChopDist schop-d-def wpow-Suc*)

lemma *PiFPowerSucb*:
$\vdash (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))$
proof −
have *0*: $\vdash (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f)^\frown((init\ w) \wedge (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*meson PiFPowerSuca*)
have *1*: $\vdash((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k)) = (w \wedge empty)^\frown\ ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*metis InitAndEmptyEqvAndEmpty Prop10 Prop12 StateAndEmptySChop int-iffD2 inteq-reflection lift-and-com*)
have *2*: $\vdash((init\ w)\ \Pi\ f)^\frown((init\ w) \wedge (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$((init\ w)\ \Pi\ f)^\frown((w \wedge empty)^\frown\ ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k)))$
using *1 RightSChopEqvSChop* **by** *blast*
have *3*: $\vdash ((init\ w)\ \Pi\ f)^\frown((w \wedge empty)^\frown\ ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))) =$
$(((init\ w)\ \Pi\ f)^\frown\ (w \wedge empty))^\frown((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*simp add*: *SChopAssoc*)
have *4*: $\vdash (((init\ w)\ \Pi\ f)^\frown\ (w \wedge empty)) = ((init\ w)\ \Pi\ f \wedge sfin\ w)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have *5*: $\vdash (((init\ w)\ \Pi\ f)^\frown\ (w \wedge empty))^\frown((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*simp add*: *4 LeftSChopEqvSChop*)
show *?thesis*
using *0 2 3 5* **by** *fastforce*
qed

lemma *PiFPowerSucc*:
$\vdash (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) =$
$(fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ k))^\frown((init\ w) \wedge (init\ w)\ \Pi\ empty)$
proof (*induction k*)
case *0*
then show *?case*
using *PiFPowerSucb*[*of w f 0*]
unfolding *schop-d-def fpower-d-def wpow-0 wpow-Suc*
proof −
assume *a1*: $\vdash init\ w\ \Pi\ ((f \wedge finite);empty) =$
$((init\ w\ \Pi\ f \wedge sfin\ w) \wedge finite);(init\ w \wedge init\ w\ \Pi\ empty)$
have $\vdash(((init\ w\ \Pi\ f \wedge sfin\ w) \wedge finite);empty \wedge finite) = ((init\ w\ \Pi\ f \wedge sfin\ w) \wedge finite)$
by (*metis AndChopB ChopEmpty Prop10 inteq-reflection*)
then show $\vdash init\ w\ \Pi\ ((f \wedge finite);empty) =$

```
          (((init w Π f ∧ sfin w) ∧ finite);empty ∧ finite);(init w ∧ init w Π empty)
  by (metis a1 int-eq)
qed
next
case (Suc k)
  then show ?case
  proof −
    have 3: ⊢ (init w Π f) ⌢
              (init w ∧ fpower (init w Π f ∧ sfin w) (Suc k) ⌢ (init w ∧ init w Π empty)) =
              ((init w Π f ∧ sfin w) ⌢ fpower (init w Π f ∧ sfin w) (Suc k)) ⌢
              (init w ∧ init w Π empty)
    by (metis (no-types, lifting) AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SChopAssoc
        SFinEqvTrueSChopAndEmpty inteq-reflection)
    have 4: ⊢(init w Π f) ⌢ (init w ∧ init w Π (f ⌢ fpower f k)) =
              ((init w Π f ∧ sfin w) ⌢
               ((init w Π f ∧ sfin w) ⌢ fpower (init w Π f ∧ sfin w) k)) ⌢
              (init w ∧ init w Π empty)
      using 3 Suc.IH unfolding fpower-d-def wpow-Suc schop-d-def by (metis int-eq)
    show ?thesis
      by (metis 4 PiFPowerSuca fpower-d-def inteq-reflection schop-d-def wpow-Suc)
  qed
qed

lemma PiFPowerSucd:
  ⊢ (init w) Π (fpower f (Suc k)) = (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) Π empty)
proof (induction k)
case 0
then show ?case unfolding fpower-d-def wpow-0 wpow-Suc
by (metis (no-types, opaque-lifting) InitAndEmptyEqvAndEmpty PiSChopDist SChopAndEmptyEqvSChopAn-
dEmpty
   SChopAssoc SFinEqvTrueSChopAndEmpty StateAndEmptySChop inteq-reflection lift-and-com schop-d-def)
next
case (Suc k)
then show ?case
  proof −
   have 1: ⊢  init w Π fpower f (Suc (Suc k)) = (init w) Π (f⌢(fpower f (Suc k)))
     by (metis PiFPowerSuca PiSChopDist inteq-reflection)
   have 2: ⊢ (init w) Π (f⌢(fpower f (Suc k))) = ((init w) Π f)⌢((init w) ∧ (init w) Π (fpower f (Suc
k)))
     by (simp add: PiSChopDist)
   have 3: ⊢ ((init w) Π f)⌢((init w) ∧ (init w) Π (fpower f (Suc k))) =
               ((init w) Π f)⌢((init w) ∧ (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) ∧ (init w) Π
empty))
     by (metis 2 PiFPowerSucc inteq-reflection)
   have 4: ⊢ ((init w) Π f)⌢((init w) ∧ (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) ∧ (init w) Π
empty)) =
                ((init w) Π f ∧ sfin w)⌢( (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) ∧ (init w) Π
empty))
        by (metis AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SFinEqvTrueSChopAn-
dEmpty inteq-reflection)
```

have *5*: $\vdash ((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w)\ \Pi\ empty)) =$
$(\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ (Suc\ k))) \frown ((init\ w) \wedge (init\ w)\ \Pi\ empty))$
by (*metis 1 2 4 PiFPowerSucc inteq-reflection*)
have *6*: $\vdash (\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ (Suc\ k))) \frown ((init\ w) \wedge (init\ w)\ \Pi\ empty)) =$
$(\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ (Suc\ k))) \frown (\ (init\ w)\ \Pi\ empty))$
by (*metis* (*no-types*, *lifting*) *PiFPowerSucc SChopAssoc Suc fpower-d-def inteq-reflection schop-d-def wpow-Suc*)
show *?thesis*
by (*metis 1 2 3 4 5 6 inteq-reflection*)
qed
qed

lemma *SFin-SChop*:
$\vdash (f \wedge sfin\ w) \frown g = (f \wedge fin\ w) \frown g$
proof −
have *1*: $\vdash (f \wedge fin\ w) \frown g\ = (f \wedge (fin\ w \wedge finite)) \frown g$
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop Prop11 Prop12 lift-and-com schop-d-def*)
have *2*: $\vdash (fin\ w \wedge finite) = (sfin\ w \wedge finite)$
by (*metis* (*no-types*, *lifting*) *EmptyImpFinite FinAndEmpty FinEqvTrueChopAndEmpty Finprop*(*5*) *Prop10 SChopAndB SChopImpFinite SFinAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-imp-trans sfin-d-def*)
have *3*: $\vdash (f \wedge sfin\ w) \frown g\ = (f \wedge (sfin\ w \wedge finite)) \frown g$
by (*metis AndSChopCommute DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond int-eq*)
show *?thesis*
using *1 2 3* **by** (*metis inteq-reflection*)
qed

lemma *PiChopstar*:
$\vdash (init\ w)\ \Pi\ (schopstar\ f) =$
$(init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge (schopstar(((init\ w)\ \Pi\ f) \wedge sfin\ w)) \frown wnext(\Box\ (init\ (\neg\ w))))$
proof −
have *1*: $\vdash (init\ w)\ \Pi\ (schopstar\ f) = (init\ w)\ \Pi\ (\exists\, k.\ fpower\ f\ k)$
by (*metis FPowerstardef PiEqvRule SChopstar-FPowerstar inteq-reflection*)
have *2*: $\vdash (init\ w)\ \Pi\ (\exists\, k.\ fpower\ f\ k) = (\exists\, k.\ (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*simp add*: *Valid-def pi-d-def*)
have *3*: $\vdash (\exists\, k.\ (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$(\ (init\ w)\ \Pi\ empty \vee (\exists\, k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))))$
using *PiFPowerExpand* **by** *auto*
have *4*: $\vdash (\exists\, k.\ (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k))) =$
$(\exists\, k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w)\ \Pi\ empty))$
by (*meson ExEqvRule PiFPowerSucd*)
have *5*: $\vdash (init\ w)\ \Pi\ empty = empty \frown ((init\ w)\ \Pi\ empty)$
by (*simp add*: *EmptySChop int-iffD1 int-iffD2 int-iffI*)
have *6*: $\vdash (\exists\, k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w)\ \Pi\ empty)) =$
$(\exists\, k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ k))) \frown ((init\ w)\ \Pi\ empty)$
by (*simp add*: *ExistSChop*)
have *7*: $\vdash (\ (init\ w)\ \Pi\ empty \vee (\exists\, k.\ (fpower\ ((init\ w)\ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w)\ \Pi\ empty))) =$

$(\ empty\frown\ ((init\ w)\ \Pi\ empty)\ \vee$
$(\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (Suc\ k)))\frown((init\ w)\ \Pi\ empty))$
using *5 6* **by** *fastforce*
have *8*: $\vdash$ $(\ empty\frown\ ((init\ w)\ \Pi\ empty)\ \vee$
$(\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (Suc\ k)))\frown((init\ w)\ \Pi\ empty)) =$
$(\ empty\ \vee\ (\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (Suc\ k))))\frown((init\ w)\ \Pi\ empty)$
by (*meson OrSChopEqv Prop11*)
have *9*: $\vdash$ $empty = (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ 0)$
unfolding *fpower-d-def* **by** *simp*
have *10*: $\vdash$ $(\ empty\ \vee\ (\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (Suc\ k)))) =$
$(\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ 0)\ \vee\ (\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (Suc\ k))))$
unfolding *fpower-d-def* **by** *simp*
have *11*: $\vdash$ $(\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ 0)\ \vee\ (\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (Suc\ k)))) =$
$(\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ k))$
using *exists-expand*[*of w f*] **unfolding** *fpower-d-def*
by (*metis* (*mono-tags*) *Valid-def inteq-reflection wpow-0 wpowersem1*)
have *12*: $\vdash$ $(\ (init\ w)\ \Pi\ empty\ \vee$
$(\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (Suc\ k))\frown((init\ w)\ \Pi\ empty))) =$
$(\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (\ k)))\frown\ ((init\ w)\ \Pi\ empty)$
by (*metis 11 7 8 9 inteq-reflection*)
have *13*: $\vdash$ $(\exists k.\ (fpower\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ (\ k)))\frown\ ((init\ w)\ \Pi\ empty) =$
$(schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w))\frown\ \ ((init\ w)\ \Pi\ empty)$
by (*metis FPowerstardef LeftSChopEqvSChop SChopstar-FPowerstar inteq-reflection*)
have *14*: $\vdash$ $(schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w))\frown\ \ ((init\ w)\ \Pi\ empty) =$
$(\ ((init\ w)\ \Pi\ empty)\ \vee$
$(((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\frown(schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)))\frown((init\ w)\ \Pi\ empty))$
by (*metis 5 OrSChopEqvRule SChopstar-unfoldl-eq inteq-reflection*)
have *15*: $\vdash$ $((init\ w)\ \Pi\ empty) =\ \ (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w)\ \wedge\ wnext\ (\Box\ (init\ (\neg\ w))))$
by (*simp add*: *PiEmpty*)
have *16*:$\vdash$ $(\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ \frown\ (schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w))\ )\frown((init\ w)\ \Pi\ empty) =$
$(\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ \frown\ (schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w))\ )\frown\ wnext\ (\Box\ (init\ (\neg\ w)))$
proof −
have *161*: $\vdash$ $(\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ \frown\ (schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w))\ )\frown((init\ w)\ \Pi\ empty) =$
$(\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ \frown\ ((schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w))\ \frown((init\ w)\ \Pi\ empty)))$
by (*meson Prop11 SChopAssoc*)
have *162*: $\vdash$ $((init\ w)\ \Pi\ f\ \wedge\ sfin\ w) = ((init\ w)\ \Pi\ f)\frown(w\ \wedge\ empty)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)

have *163*: $\vdash$ $(schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)) = (schopstar\ (((init\ w)\ \Pi\ f)\frown\ (w\ \wedge\ empty)))$
by (*metis 162 SChopstardef inteq-reflection*)
have *164*: $\vdash$ $((init\ w)\ \Pi\ f\ \wedge\ sfin\ w)\ \frown\ (schopstar\ ((init\ w)\ \Pi\ f\ \wedge\ sfin\ w))\ =$
$(\ (init\ w)\ \Pi\ f)\frown((w\ \wedge\ empty)\frown(schopstar\ (\ ((init\ w)\ \Pi\ f)\ \frown\ (w\ \wedge\ empty))))$
using *162 SChopAssoc int-eq* **by** *metis*
have *165*: $\vdash$ $(\ (init\ w)\ \Pi\ f)\frown((w\ \wedge\ empty)\frown(schopstar\ (\ ((init\ w)\ \Pi\ f)\ \frown\ (w\ \wedge\ empty)))) =$
$(\ (init\ w)\ \Pi\ f)\frown\ ((schopstar\ (\ (w\ \wedge\ empty)\frown\ (\ (init\ w)\ \Pi\ f)))\frown(w\ \wedge\ empty))$
by (*simp add*: *PiChopstarhelp2 RightSChopEqvSChop*)
have *166*: $\vdash$ $((\ (init\ w)\ \Pi\ f)\frown\ ((schopstar\ (\ (w\ \wedge\ empty)\frown\ (\ (init\ w)\ \Pi\ f)))\frown(w\ \wedge\ empty)))\frown((init\ w)\ \Pi\ empty) =$
$(\ ((\ (init\ w)\ \Pi\ f)\frown\ (schopstar\ (\ (w\ \wedge\ empty)\frown\ (\ (init\ w)\ \Pi\ f))))\ \frown(\ (init\ w)\ \wedge\ ((init\ w)\ \Pi\ empty)\ )\ )$

by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SChopAssoc StateAndEmptySChop int-eq*)
have *167*: $\vdash$ ((((*init w*) Π *f*)$\frown$ (*schopstar* ((*w* $\wedge$ *empty*)$\frown$ ((*init w*) Π *f*)))) $\frown$((*init w*) $\wedge$ ((*init w*) Π *empty*))) =
((((*init w*) Π *f*)$\frown$ (*schopstar* ((*w* $\wedge$ *empty*)$\frown$ ((*init w*) Π *f*)))) $\frown$((*w* $\wedge$ *empty*) $\frown$ *wnext* ($\Box$ (*init* ($\neg$ *w*)))))
by (*simp add*: *RightSChopEqvSChop StateAndPiEmpty*)
have *168*: $\vdash$ ((((*init w*) Π *f*)$\frown$ (*schopstar* ((*w* $\wedge$ *empty*)$\frown$ ((*init w*) Π *f*)))) $\frown$((*w* $\wedge$ *empty*) $\frown$ *wnext* ($\Box$ (*init* ($\neg$ *w*))))) =
((((*init w*) Π *f*)$\frown$ (*schopstar* ((*w* $\wedge$ *empty*)$\frown$ ((*init w*) Π *f*)))) $\frown$(*w* $\wedge$ *empty*)) $\frown$ *wnext* ($\Box$ (*init* ($\neg$ *w*)))
by (*simp add*: *SChopAssoc*)
have *169*: $\vdash$ ((((*init w*) Π *f*)$\frown$ (*schopstar* ((*w* $\wedge$ *empty*)$\frown$ ((*init w*) Π *f*)))) $\frown$(*w* $\wedge$ *empty*)) =
(((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)))
using *164 165 SChopAssoc* **by** *fastforce*
show *?thesis* **by** (*metis 164 165 166 167 168 169 int-eq*)
qed
have *17*: $\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*) = ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ *empty*)
by (*metis InitAndEmptyEqvAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty*
inteq-reflection lift-and-com)
have *18*: $\vdash$ ((*init w*) Π *f*) = *wprev*($\Box$ (*init* ($\neg$ *w*)))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*)
by (*simp add*: *WPrevPi*)
have *19*: $\vdash$ *wprev*($\Box$ (*init* ($\neg$ *w*)))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*) =
wprev($\Box$ (*init* ($\neg$ *w*)))$\frown$(((*init w*) $\wedge$ *empty*) $\frown$ ((*init w*) Π *f*))
by (*meson RightSChopImpSChop StateAndEmptySChop int-iffD1 int-iffD2 int-iffI*)
have *20*: $\vdash$ *wprev*($\Box$ (*init* ($\neg$ *w*)))$\frown$(((*init w*) $\wedge$ *empty*) $\frown$ ((*init w*) Π *f*)) =
(*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ ((*init w*) Π *f*))
using *18 19 StatePiUntil* **by** *fastforce*
have *21*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)))$\frown$ *wnext* ($\Box$ (*init* ($\neg$ *w*))) =
(*init* ($\neg$ *w*)) $\mathcal{U}$ (
(*init w*) $\wedge$
((((*init w*) Π *f*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)))$\frown$*wnext* ($\Box$ (*init* ($\neg$ *w*)))
)
proof $-$
have $\vdash$(*init w* Π *f*) $\frown$ (*init w* $\wedge$ *empty*) = (*init w* Π *f* $\wedge$ *sfin w*)
by (*metis 17 inteq-reflection*)
then show *?thesis* **using** *StatePiUntil*[*of w f*]
UntilChopDist[*of w LIFT*((*init w* $\wedge$ *empty*)) *LIFT*((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)))$\frown$*wnext* ($\Box$ (*init* ($\neg$ *w*)))]
by (*metis* (*no-types*, *lifting*) *AndSFinSChopEqvStateAndSChop SChopAssoc StateUntilEqvWPrevChop int-eq*)
qed
have *22*: $\vdash$ ((*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*)))) $\vee$
(*init* ($\neg$ *w*)) $\mathcal{U}$ (
(*init w*) $\wedge$
((((*init w*) Π *f*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)))$\frown$*wnext* ($\Box$ (*init* ($\neg$ *w*)))
)) =
(*init* ($\neg$ *w*)) $\mathcal{U}$ (
((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*)))) $\vee$
((*init w*) $\wedge$ ((((*init w*) Π *f*) $\wedge$ *sfin w*)$\frown$(*schopstar*((*init w*) Π *f* $\wedge$ *sfin w*)))$\frown$*wnext* ($\Box$ (*init* ($\neg$ *w*))))

```
                                )
  using UntilOrDist by fastforce
 have 23: ⊢ (
       ((init w) ∧ wnext (□ (init (¬ w)))) ∨
      ( (init w) ∧ ((((init w) Π f) ∧ sfin w)⌢(schopstar ((init w) Π f ∧ sfin w)))⌢wnext (□ (init (¬ w))))
           ) =
           ( (init w) ∧ (schopstar (((init w) Π f) ∧ sfin w))⌢wnext(□ (init (¬ w))))
   by (metis (mono-tags, lifting) EmptyOrSChopEqv Prop11 SChopOrEqvRule SChopstar-unfoldl-eq State-
AndEmptySChop int-eq)
 have 24: ⊢ (init w) Π (schopstar f) = ( (init w) Π empty ∨ (∃ k. (init w) Π (fpower f (Suc k))))
  using 1 2 3 by fastforce
 have 25: ⊢ ( (init w) Π empty ∨ (∃ k. (init w) Π (fpower f (Suc k)))) =
           ( (init w) Π empty ∨ (∃ k. (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) Π empty)))
  using 4 by fastforce
 have 26: ⊢ ( (init w) Π empty ∨
             (∃ k. (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) Π empty))) =
           (schopstar ((init w) Π f ∧ sfin w)) ⌢  ((init w) Π empty)
  using 12 13 by fastforce
 have 27: ⊢ (schopstar ((init w) Π f ∧ sfin w))⌢  ((init w) Π empty) =
         ((init (¬ w)) 𝒰 ( (init w) ∧ wnext (□ (init (¬ w)))) ∨
          ( ((init w) Π f ∧ sfin w) ⌢ (schopstar ((init w) Π f ∧ sfin w)) )⌢ wnext (□ (init (¬ w))))
  using 14 15 16 by fastforce
 have 28:⊢ (schopstar ((init w) Π f ∧ sfin w)) ⌢  ((init w) Π empty) =
          (init (¬ w)) 𝒰 ((init w) ∧ (schopstar (((init w) Π f) ∧ sfin w))⌢wnext(□ (init (¬ w))))
  using 27  21 22 23
  by (metis inteq-reflection)
 show ?thesis
  by (metis 24 25 26 28 inteq-reflection)
qed


lemma PiChopstar-var1:
  ⊢ (init w) Π (schopstar f) =
     (schopstar ((init w) Π f ∧ sfin w)) ⌢  ((init w) Π empty)
proof −
 have 1: ⊢ (init w) Π (schopstar f) = (init w) Π (∃ k. fpower f k)
   by (metis FPowerstardef PiEqvRule SChopstar-FPowerstar inteq-reflection)
 have 2: ⊢ (init w) Π (∃ k. fpower f k) = (∃ k. (init w) Π (fpower f k))
  by (simp add: Valid-def pi-d-def)
 have 3: ⊢ (∃ k. (init w) Π (fpower f k)) =
           ( (init w) Π empty ∨ (∃ k. (init w) Π (fpower f (Suc k))))
  using PiFPowerExpand by auto
 have 4: ⊢ (∃ k. (init w) Π (fpower f (Suc k))) =
           (∃ k. (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) Π empty))
  by (meson ExEqvRule PiFPowerSucd)
 have 5: ⊢ (init w) Π empty = empty ⌢ ((init w) Π empty)
  by (simp add: EmptySChop int-iffD1 int-iffD2 int-iffI)
 have 6: ⊢ (∃ k. (fpower ((init w) Π f ∧ sfin w) (Suc k))⌢((init w) Π empty)) =
           (∃ k. (fpower ((init w) Π f ∧ sfin w) (Suc k)))⌢((init w) Π empty)
  by (simp add: ExistSChop)
```

have *7*: $\vdash$ ((*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) Π *empty*)))=
(*empty*$\frown$ ((*init w*) Π *empty*) $\vee$
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))$\frown$((*init w*) Π *empty*))
using *5 6* **by** *fastforce*
have *8*: $\vdash$ (*empty*$\frown$ ((*init w*) Π *empty*) $\vee$
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))$\frown$((*init w*) Π *empty*)) =
(*empty* $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))))$\frown$((*init w*) Π *empty*)
by (*meson OrSChopEqv Prop11*)
have *9*: $\vdash$ *empty* = (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *0*)
unfolding *fpower-d-def* **by** *simp*
have *10*: $\vdash$ (*empty* $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))) =
((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *0*) $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))))
unfolding *fpower-d-def* **by** *simp*
have *11*: $\vdash$ ((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *0*) $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))) =
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *k*))
using *exists-expand*[*of w f*] **unfolding** *fpower-d-def*
by (*metis* (*mono-tags*) *Valid-def inteq-reflection wpow-0 wpowersem1*)
have *12*: $\vdash$ ((*init w*) Π *empty* $\vee$
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) Π *empty*))) =
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*k*)))$\frown$ ((*init w*) Π *empty*)
by (*metis 11 7 8 9 inteq-reflection*)
have *13*: $\vdash$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*k*)))$\frown$ ((*init w*) Π *empty*) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)
by (*metis FPowerstardef LeftSChopEqvSChop SChopstar-FPowerstar inteq-reflection*)
show *?thesis*
by (*metis 1 12 13 2 3 4 inteq-reflection*)
qed

lemma *PiChopstar-var2*:
$\vdash$ (*init w*) Π (*schopstar f*) =
((*init w*) Π *empty* $\vee$
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$(((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$(*wnext* (□ (*init* (¬*w*))))))

proof −
have *1*: $\vdash$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) =
(*empty* $\vee$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ ((*init w*) Π *f* $\wedge$ *sfin w*))
by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite Prop10 Prop12 SCSEqvOrChopSCSB SChopAndB*

SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond int-eq int-iffD2 lift-and-com)
have *2*: $\vdash$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*) =
(((*init w*) Π *empty*) $\vee$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$((*init w*) Π *empty*))
using *1 EmptyOrSChopEqvRule* **by** *blast*
have *3*: $\vdash$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$((*init w*) Π *empty*) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ (((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *empty*))
by (*meson Prop11 SChopAssoc*)
have *4*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *empty*)) =
(((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$(*wnext* (□ (*init* (¬*w*)))))
by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAndPiEmpty inteq-reflection*)

have *5*: $\vdash$ $(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))^\frown\ (((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown((init\ w)\ \Pi\ empty)) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))^\frown(((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown(wnext\ (\Box\ (init\ (\neg w)))))$
by (*metis 4 RightChopEqvChop schop-d-def*)
show *?thesis*
by (*metis 2 3 5 PiChopstar-var1 int-eq*)
qed

lemma *PiChopstar-var3*:
$\vdash (init\ w)\ \Pi\ (f^\frown(schopstar\ f)\ ) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))\ ^\frown(((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown(wnext\ (\Box\ (init\ (\neg w)))))$
proof −
have *1*: $\vdash (init\ w)\ \Pi\ (f^\frown(schopstar\ f)) =$
$((init\ w)\ \Pi\ f \wedge sfin\ w)\ ^\frown\ (\ (init\ w)\ \Pi\ (schopstar\ f))$
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)
have *2*: $\vdash ((init\ w)\ \Pi\ (schopstar\ f)) = (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))\ ^\frown\ \ ((init\ w)\ \Pi\ empty)$
using *PiChopstar-var1* **by** *auto*
have *3*: $\vdash ((init\ w)\ \Pi\ f \wedge sfin\ w)\ ^\frown\ (\ (init\ w)\ \Pi\ (schopstar\ f)) =$
$((init\ w)\ \Pi\ f \wedge sfin\ w)\ ^\frown((schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))\ ^\frown\ \ ((init\ w)\ \Pi\ empty))$
using *2 RightSChopEqvSChop* **by** *blast*
have *4*: $\vdash ((init\ w)\ \Pi\ f \wedge sfin\ w)\ ^\frown((schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))\ ^\frown\ \ ((init\ w)\ \Pi\ empty)) =$
$(((init\ w)\ \Pi\ f \wedge sfin\ w)\ ^\frown(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)))\ ^\frown\ \ ((init\ w)\ \Pi\ empty)$
by (*simp add*: *SChopAssoc*)
have *5*: $\vdash$ $(((init\ w)\ \Pi\ f \wedge sfin\ w)\ ^\frown(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))^\frown\ (((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite)$
by (*simp add*: *SChopplusCommute*)
have *6*: $\vdash (((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite) = (((init\ w)\ \Pi\ f \wedge sfin\ w))$
using *SFinEQvFinAndFinite* **by** *fastforce*
have *7*: $\vdash (((init\ w)\ \Pi\ f \wedge sfin\ w))^\frown((init\ w)\ \Pi\ empty) =$
$((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown\ (wnext\ (\Box\ (init\ (\neg w))))$
by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAndPiEmpty inteq-reflection*)
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 2 5 6 7 SChopAssoc inteq-reflection*)
qed

lemma *PiChopstar-var4*:
$\vdash (init\ w)\ \Pi\ (f^\frown(schopstar\ f)\ ) =$
$(((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown\ (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)))\ ^\frown(wnext\ (\Box\ (init\ (\neg w))))$
proof −
have *1*: $\vdash$ $(init\ w)\ \Pi\ (f^\frown(schopstar\ f)\ ) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))\ ^\frown(((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown(wnext\ (\Box\ (init\ (\neg w)))))$
using *PiChopstar-var3* **by** *blast*
have *2*: $\vdash (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))\ ^\frown(((init\ w)\ \Pi\ f \wedge sfin\ w)^\frown(wnext\ (\Box\ (init\ (\neg w))))) =$
$((schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))\ ^\frown((init\ w)\ \Pi\ f \wedge sfin\ w))^\frown(wnext\ (\Box\ (init\ (\neg w))))$
by (*simp add*: *SChopAssoc*)
have *3*: $\vdash ((init\ w)\ \Pi\ f \wedge sfin\ w) = (((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite)$
using *SFinEQvFinAndFinite* **by** *fastforce*

have *4*: $\vdash$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ (((*init w*) Π *f* $\wedge$ *sfin w*) $\wedge$ *finite*) =
(((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)))
by (*simp add*: *SChopstar-slide-var*)
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *PiChopstar-var5*:
$\vdash$ (*init w*) Π (*f*$\frown$(*schopstar f*)) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$((*init w*) Π (*f* $\wedge$ *finite*))
proof −
have *1*: $\vdash$ (*init w*) Π (*f*$\frown$(*schopstar f*)) =
((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ ((*init w*) Π (*schopstar f*))
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)
have *2*: $\vdash$ ((*init w*) Π (*schopstar f*)) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)
using *PiChopstar-var1* **by** *auto*
have *3*: $\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ ((*init w*) Π (*schopstar f*)) =
((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*))
using *2 RightSChopEqvSChop* **by** *blast*
have *4*: $\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)) =
(((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*)
using *SChopAssoc* **by** *blast*
have *5*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ (((*init w*) Π *f* $\wedge$ *sfin w*) $\wedge$ *finite*)
by (*simp add*: *SChopplusCommute*)
have *6*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*) $\wedge$ *finite*) = (((*init w*) Π *f* $\wedge$ *sfin w*))
using *SFinEQvFinAndFinite* **by** *fastforce*
have *7*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*) =
(*init w*) Π (*f*$\frown$*empty*)
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)
have *8*: $\vdash$ (*init w*) Π (*f*$\frown$*empty*) = (*init w*) Π (*f* $\wedge$ *finite*)
by (*simp add*: *ChopEmpty PiEqvRule schop-d-def*)
have *9*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*) =
((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$(((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*)
by (*metis 4 5 6 inteq-reflection*)
have *10*: $\vdash$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$(((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$((((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*))
by (*meson Prop11 SChopAssoc*)
show *?thesis*
by (*metis 1 10 3 4 7 8 9 int-eq*)
qed

lemma *PiChopstar-var6*:
$\vdash$ (*init w*) Π (*schopstar f*) =
((((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))
$\frown$(*wnext*($\Box$(*init* ($\neg$*w*)))))
proof −

have *1*: $\vdash$ (*schopstar f*) = (*schopstar* (*f* $\vee$ *empty*))
by (*metis SChopstar-star2 SChopstar-swap inteq-reflection*)
have *1*: $\vdash$ (*init w*) Π (*schopstar* (*f* $\vee$ *empty*)) =
(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)
by (*simp add*: *PiChopstar-var1*)
have *2*: $\vdash$ (*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) =
(*empty* $\vee$ ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))
by (*meson Prop06 SCSEqvOrChopSCSB SChopstar-slide-var*)
have *3*: $\vdash$ (*empty* $\vee$ ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*) Π *empty*)
= (((*init w*) Π *empty*) $\vee$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*) Π *empty*))

using *EmptyOrSChopEqv* **by** *blast*
have *4*: $\vdash$ ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) =
(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*))$\frown$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*) $\wedge$ *finite*)
by (*simp add*: *SChopplusCommute*)
have *5*: $\vdash$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*) $\wedge$ *finite*) =
(((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*))
using *SFinEQvFinAndFinite* **by** *fastforce*
have *6*: $\vdash$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*) =
(((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) $\frown$ (*wnext*($\Box$(*init* ($\neg$*w*))))
by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAndPiEmpty int-eq*)
have *7*: $\vdash$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*) Π *empty*) =
(((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ (*wnext*($\Box$(*init* ($\neg$*w*))))
by (*metis* (*no-types*, *lifting*) *4 5 6 SChopAssoc inteq-reflection*)
have *8*: $\vdash$ (((*init w*) Π *empty*) $\vee$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*) Π *empty*)) =
((((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*) Π *empty*))
by (*metis 1 2 3 7 FiniteAndEmptyEqvEmpty PiChopstar-var4 Prop05 SChopstar-sup-id-star1 int-iffD1 inteq-reflection*)
show *?thesis*
by (*meson PiChopstar-var4 PiEqvRule Prop04 SChopstar-star2 SChopstar-sup-id-star1 SChopstar-swap UntilImpOr UntilIntro lift-imp-trans*)
qed

3.9.6 Omega and Pi

lemma *OmegaImpDiamond*:
$\vdash$ (*omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*)) $\longrightarrow$ $\Diamond$ (*init w*)
proof −
have *1*: $\vdash$ (*omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*)) =
((((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *more*) $\wedge$ *finite*);(*omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*))
using *OmegaUnroll*[*of LIFT* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*)]
by (*simp add*: *schop-d-def*)
have *2*: $\vdash$ ((((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *more*) $\wedge$ *finite*) $\longrightarrow$ $\Diamond$ (*init w*)

using *PiImpDiamond* **by** *fastforce*
have *3*: $\vdash$ ((((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *more*) $\wedge$ *finite*) =
(((((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*)
by *auto*
have *4*: $\vdash$ ((((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *more*) $\wedge$ *finite*);(*omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*))
=
((((((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);(*omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*))
using *3 LeftChopEqvChop* **by** *blast*
have *5*: $\vdash$ ((((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);(*omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*))
= (((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ *omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*))
using *SlideInitSFin*[*of LIFT* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *more*) *w*
LIFT (*omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*))] **unfolding** *schop-d-def*
by (*metis inteq-reflection*)
have *6*: $\vdash$ (((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ *omega* ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *sfin w*))
$\longrightarrow$ $\Diamond$ (*init w*)
by (*metis ChopAndB FiniteChopImpDiamond inteq-reflection lift-and-com lift-imp-trans*)
show *?thesis* **using** *1 3 4 5 6* **by** (*metis int-eq*)
qed

lemma *PiStateOmegaUnroll*:
$\vdash$ (*init w*) Π (*omega f*) = (((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *sfin w*); ((*init w*) Π (*omega f*))
proof −
have *1*: $\vdash$ (*init w*) Π (*omega f*) = (*init w*) Π (((*f* $\wedge$ *more*) $\wedge$ *finite*);(*omega f*))
by (*metis OmegaUnroll PiEqvRule schop-d-def*)
have *2*: $\vdash$ (*init w*) Π (((*f* $\wedge$ *more*) $\wedge$ *finite*);(*omega f*)) =
((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*))
using *PiSChopDist*[*of w LIFT* (*f* $\wedge$ *more*) *LIFT* (*omega f*)] **unfolding** *schop-d-def*
by *simp*
have *3*: $\vdash$ ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *finite*) = ((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*)
by (*meson PiFiniteAbsorb Prop11*)
have *4*: $\vdash$ ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*)) =
((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*))
by (*simp add*: *3 LeftChopEqvChop*)
have *5*: $\vdash$ ((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*)) =
(((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π (*omega f*))
using *SlideInitSFin*[*of LIFT* (*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) *w LIFT* (*init w*) Π (*omega f*)]
unfolding *schop-d-def* **by** *blast*
have *6*: $\vdash$ (((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*) $\wedge$ *finite*) =
(((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*))
using *SFinEQvFinAndFinite* **by** *fastforce*
have *7*: $\vdash$ (*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) = ((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *more*)
by (*metis PiAnd PiAndPiImpPiAnd PiMoreAbsorb Prop10 Prop12 inteq-reflection*)
have *8*: $\vdash$ (((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*)) =
((((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *sfin w*))
using *7* **by** *auto*
show *?thesis*
by (*metis 1 2 4 5 6 8 int-eq*)
qed

lemma *PiAOmega-sfin*:
assumes ¬ *nfinite l*
nnth l 0 = 0
nidx l
(∀ *i*. *enat* (*nnth l i*) ≤ *nlength σ*)
shows ((*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *sfin w*) =
w (*NNil* (*nnth σ* (*nnth l* (*Suc i*))))
using *assms nidx-expand*[*of l*]
by (*simp add*: *sfin-defs*)
(*metis diff-add less-imp-le-nat linorder-le-cases ndropn-nlast nfinite-ntaken nsubn-def1 ntaken-all ntaken-ndropn-nlast*)

lemma *PiAOmega-sfin-exist*:
assumes ¬ *nfinite l*
nnth l 0 = 0
nidx l
(∀ *i*. *enat* (*nnth l i*) ≤ *nlength σ*)
w (*NNil* (*nnth σ* (*nnth l* (*Suc i*))))
shows (∃ *x*∈*nset* (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))) . *w* (*NNil x*))
using *assms nidx-expand*[*of l*] *in-nset-conv-nnth*[*of* - (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)))]
unfolding *nsubn-def1*
by (*metis diff-add less-imp-le-nat linorder-le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-ndropn-nlast*)

lemma *PiAOmega-help2*:
(*σ* ⊨ (*aomega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*))) =
(∃ *l*. ¬ *nfinite l* ∧
nnth l 0 = 0 ∧
nidx l ∧ (∀ *i*. *enat* (*nnth l i*) ≤ *nlength σ*) ∧
(∀ *i*. (
((*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)))) ⊨ *f*) ∧
0< *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)))) ∧
nfinite (*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))
∧ *w* (*NNil* (*nnth σ* (*nnth l* (*Suc i*))))
)
))

proof −
have *1*: (*σ* ⊨ (*aomega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*))) =
(∃ *l*. ¬ *nfinite l* ∧
nnth l 0 = 0 ∧
nidx l ∧ (∀ *i*. *enat* (*nnth l i*) ≤ *nlength σ*) ∧
(∀ *i*. *nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *init w* Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*))
unfolding *aomega-d-def* **by** *blast*
have *2*: (∃ *l*. ¬ *nfinite l* ∧
nnth l 0 = 0 ∧
nidx l ∧ (∀ *i*. *enat* (*nnth l i*) ≤ *nlength σ*) ∧
(∀ *i*. *nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *init w* Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*)) =

$(\exists\, l.\ \neg\ nfinite\ l\ \wedge$
$nnth\ l\ 0 = 0\ \wedge$
$nidx\ l\ \wedge\ (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma)\ \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x))\ \wedge$
$(nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models (f\ \wedge\ more)\ \wedge\ finite)$
$\wedge\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w)$
$)$
$)\ )$
using *Pistate*[*of w LIFT* $((f\ \wedge\ more)\ \wedge\ finite)$] **by** *auto*
have *3*: $(\exists\, l.\ \neg\ nfinite\ l\ \wedge$
$nnth\ l\ 0 = 0\ \wedge$
$nidx\ l\ \wedge\ (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma)\ \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x))\ \wedge$
$(nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models (f\ \wedge\ more)\ \wedge\ finite)$
$\wedge\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w)$
$)$
$)\ ) =$
$(\exists\, l.\ \neg\ nfinite\ l\ \wedge$
$nnth\ l\ 0 = 0\ \wedge$
$nidx\ l\ \wedge\ (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma)\ \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x))\ \wedge$
$((nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models f)\ \wedge$
$0{<}\ nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))\ \wedge$
$nfinite\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
$\wedge\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w)$
$)$
$)\ )$
unfolding *more-defs finite-defs* **by** *simp*
have *4*: $(\exists\, l.\ \neg\ nfinite\ l\ \wedge$
$nnth\ l\ 0 = 0\ \wedge$
$nidx\ l\ \wedge\ (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma)\ \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x))\ \wedge$
$((nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models f)\ \wedge$
$0{<}\ nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))\ \wedge$
$nfinite\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
$\wedge\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w)$
$)$
$)\ ) =$
$(\exists\, l.\ \neg\ nfinite\ l\ \wedge$
$nnth\ l\ 0 = 0\ \wedge$
$nidx\ l\ \wedge\ (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma)\ \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x))\ \wedge$
$((nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models f)\ \wedge$
$0{<}\ nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))\ \wedge$
$nfinite\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
$\wedge\ w\ (NNil\ (nnth\ \sigma\ (nnth\ l\ (Suc\ i))))$
$)$
$)\ )$
using *PiAOmega-sfin*[*of - σ w*] **by** *blast*
have *5*: $(\exists\, l.\ \neg\ nfinite\ l\ \wedge$

```
        nnth l 0 = 0 ∧
        nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ) ∧
        (∀ i.  ( (∃ x∈nset (nsubn σ (nnth l i) (nnth l (Suc i))) . w (NNil x)) ∧
                 ((nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ⊨ f) ∧
                 0< nlength (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ∧
                 nfinite (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i))))
                 ∧ w (NNil (nnth σ (nnth l (Suc i))))
               )
        ) ) =
        (∃ l. ¬ nfinite l ∧
        nnth l 0 = 0 ∧
        nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ) ∧
        (∀ i.  (
                 ((nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ⊨ f) ∧
                 0< nlength (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ∧
                 nfinite (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i))))
                 ∧ w (NNil (nnth σ (nnth l (Suc i))))
               )
        ) )
    using PiAOmega-sfin-exist[of - σ w] by blast
 show ?thesis
 using 1 2 3 4 5 by presburger
qed


primcorec oml :: ('a ⇒ bool) ⇒ 'a nellist ⇒ nat nellist ⇒ nat ⇒ nat nellist
 where
   oml P xs l x =
     NCons (case x of 0 ⇒ 0 |
                Suc j ⇒ (the-enat (nlength(nfilter P (ntaken (nnth l x) xs)))))
         (oml P xs l (Suc x))


lemma oml-nnth-zero:
  nnth (oml P xs l 0) 0 = 0
using oml.disc-iff oml.simps(2) nhd-conv-nnth
by (metis old.nat.simps(4))

lemma oml-nnth-one:
assumes ¬ nfinite l
        ¬ nfinite xs
shows  nnth (oml P xs l 0) 1 = (the-enat (nlength(nfilter P (ntaken (nnth l 1) xs))))
using assms
oml.code oml.disc-iff oml.simps(2) nhd-conv-nnth nnth-Suc-NCons
by (metis One-nat-def old.nat.simps(5))

lemma oml-nnth:
assumes ¬ nfinite l
        ¬ nfinite xs
shows  nnth (oml P xs l x) i =
```

```
        (if x = 0 then
         (if i = 0 then 0 else (the-enat (nlength(nfilter P (ntaken (nnth l i) xs)))))
         else (the-enat (nlength(nfilter P (ntaken (nnth l (i+x)) xs)))))
using assms
proof (induct i arbitrary: x)
case 0
then show ?case
  by (metis Nitpick.case-nat-unfold ndropn-0 ndropn-nnth nhd-conv-nnth oml.disc-iff oml.simps(2))
next
case (Suc i)
then show ?case
 by (metis One-nat-def Zero-not-Suc add.commute add-Suc-shift nnth-Suc-NCons oml.code plus-1-eq-Suc)
qed

lemma oml-infinite:
 assumes ¬ nfinite l
        ¬ nfinite xs
shows  ¬ nfinite (oml P xs l x)
proof
 assume  nfinite (oml P xs l x)
 thus False
  using assms
   proof (induct zs≡ (oml P xs l x)  arbitrary: x rule: nfinite-induct)
   case (NNil y)
   then show ?case by (metis nellist.disc(1) oml.disc-iff)
   next
   case (NCons x nell)
   then show ?case by (metis nellist.sel(5) oml.simps(3))
   qed
qed


lemma PiAOmegaImpSem:
 assumes (σ ⊨ (aomega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w)))
 shows  (σ ⊨ (init w) Π (aomega f))
proof −
 have 1: (∃ l. ¬ nfinite l ∧
        nnth l 0 = 0 ∧
        nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ) ∧
        (∀ i.  (
                ((nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ⊨ f) ∧
                0< nlength (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ∧
                nfinite (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i))))
                ∧ w (NNil (nnth σ (nnth l (Suc i))))
              )
        ) )
  using assms  PiAOmega-help2[of w f σ] by blast
 obtain l where 2: ¬ nfinite l ∧
        nnth l 0 = 0 ∧
        nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ) ∧
```

```
      (∀ i.  (
              ((nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ⊨ f) ∧
              0< nlength (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ∧
              nfinite (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i))))
              ∧ w (NNil (nnth σ (nnth l (Suc i))))
             )
      )
  using 1 by auto
have 3: ⋀j. 0<j ⟶ w  (NNil (nnth σ (nnth l j)))
  using 2 by (metis Suc-diff-1)
have 31: ⋀j. 0<j ⟶ nfinite (nfilter (λy. w (NNil y)) (ntaken (nnth l j) σ))
   by (metis 2 3 nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast)
have 32: ⋀j. (nnth l j) < (nnth l (Suc j))
   by (metis 2 linorder-le-cases nfinite-ntaken nidx-expand ntaken-all)
have 4: ⋀j. 0<j ⟶
            (nfilter (λy. w (NNil y)) (nsubn σ (nnth l j) (nnth l (Suc j)))) =
            (nsubn (nfilter (λy. w (NNil y)) σ )
                   (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l j) σ))))
                   (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l (Suc j)) σ))))
            )

  using 2 3 32 nfilter-nsubn[of - σ (λy. w (NNil y))]
  by (metis 31 less-imp-le-nat nfinite-nlength-enat the-enat.simps zero-less-Suc)
have 5: (nfilter (λy. w (NNil y)) (nsubn σ 0 (nnth l (Suc 0)))) =
        (nsubn (nfilter (λy. w (NNil y)) σ )
                   0
                   (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l (Suc 0)) σ))))
            )
  by (simp add: 2 nfilter-nsubn-zero)
have 6: 0< (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l (Suc 0)) σ))))
 by (metis 2 5 gr0I i0-less nlength-NNil nsubn-def1 ntaken-0 zero-less-diff)
have 7: ⋀j. 0<j ⟶ (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l j) σ)))) <
                    (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l (Suc j)) σ))))
  by (metis 2 4 bot-nat-0.not-eq-extremum i0-less nlength-NNil nsubn-def1 ntaken-0 zero-less-diff)
have 8: ¬ nfinite (oml (λy. w (NNil y)) σ l 0)
  using 2 infinite-nidx-imp-infinite-interval oml-infinite by blast
have 9: nnth (oml (λy. w (NNil y)) σ l 0) 0 = 0
   by (simp add: oml-nnth-zero)
have 10: ⋀j. 0< j ⟶ nnth (oml (λy. w (NNil y)) σ l 0) j =
             (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l j) σ))))
  by (metis 2 infinite-nidx-imp-infinite-interval less-not-refl2 oml-nnth)
have 11: ⋀j . nnth (oml (λy. w (NNil y)) σ l 0) j < nnth (oml (λy. w (NNil y)) σ l 0) (Suc j)
  by (metis 10 6 7 9 bot-nat-0.not-eq-extremum zero-less-Suc)
have 12: nidx (oml (λy. w (NNil y)) σ l 0)
  using nidx-expand[of (oml (λy. w (NNil y)) σ l 0)]
  using 11 by blast
have 13: (∀ i. enat (nnth (oml (λy. w (NNil y)) σ l 0) i) ≤ nlength (nfilter (λy. w (NNil y)) σ))
   by (metis 10 3 bot-nat-0.not-eq-extremum enat-ile le-zero-eq linorder-le-cases
      nfilter-chop1-ntaken ntaken-all ntaken-nlast oml-nnth-zero the-enat.simps zero-enat-def)
have 14: (∀ i. f (nsubn (nfilter (λy. w (NNil y)) σ)
```

(*nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l* 0) *i*)
(*nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l* 0) (*Suc i*))))
by (*metis 10 2 4 5 9 bot-nat-0.not-eq-extremum zero-less-Suc*)
have *15*: (∃ *x*∈*nset* σ. *w* (*NNil x*))
by (*meson 2 in-nset-conv-nnth*)
have *16*: ((∃ *x*∈*nset* σ. *w* (*NNil x*)) ∧
(∃ *l*. ¬ *nfinite l* ∧
nnth l 0 = 0 ∧
nidx l ∧
(∀ *i*. *enat* (*nnth l i*) ≤ *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) σ)) ∧
(∀ *i*. *f* (*nsubn* (*nfilter* (λ*y*. *w* (*NNil y*)) σ) (*nnth l i*) (*nnth l* (*Suc i*))))))
using *12 13 14 15 8 oml-nnth-zero* **by** *blast*
have *17*: (σ ⊨ (*init w*) Π (*aomega f*))
using *16*
using *Pistate*[*of w LIFT aomega f* σ] **unfolding** *aomega-d-def* **by** *blast*
show *?thesis* **using** *17* **by** *auto*
qed

lemma *PiAOmegaImp*:
⊢ (*aomega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*)) ⟶ (*init w*) Π (*aomega f*)
unfolding *Valid-def* **using** *PiAOmegaImpSem*
using *unl-lift2* **by** *blast*

lemma *PiOmegaImpSem*:
assumes (σ ⊨ (*omega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*)))
shows (σ ⊨ (*init w*) Π (*omega f*))
by (*metis OmegaEqvAOmega PiAOmegaImpSem assms inteq-reflection*)

lemma *PiOmegaImp*:
⊢ (*omega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*)) ⟶ (*init w*) Π (*omega f*)
unfolding *Valid-def* **using** *PiOmegaImpSem* **using** *unl-lift2* **by** *blast*

lemma *PiOmega*:
⊢ (*init w*) Π (*omega f*) = (*omega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*))
proof −
have *0*: ⊢ (((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *more*) ∧ *sfin w*) =
((((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*) ∧ *more*) ∧ *finite*)
using *SFinEQvFinAndFinite* **by** *fastforce*
have *1*: ⊢ (*init w*) Π (*omega f*) ⟶ *inf*
by (*meson PiImpRule PiInfImpInf lift-imp-trans omega-imp-inf*)
have *2*: ⊢ (*init w*) Π (*omega f*) ⟶
inf ∧ (((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*) ∧ *fmore*);((*init w*) Π (*omega f*))
by (*metis 1 AndMoreSChopEqvAndFmoreChop 0 PiStateOmegaUnroll Prop12 int-eq int-iffD1 schop-d-def*)
have *3*: ⊢ (*init w*) Π (*omega f*) ⟶ (*omega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*))
using *OmegaIntro* **by** (*metis 2 Prop12*)
have *4*: ⊢ (*omega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*)) ⟶ (*init w*) Π (*omega f*)
by (*simp add*: *PiOmegaImp*)
show *?thesis*

by (*simp add*: *3 4 int-iffI*)
qed

end

3.10 Quantifiers

theory *FOTheorems*
imports
SChopTheorems Chopstar
begin

We give the proofs of a list of first order (in)finite ITL theorems.

lemma *EExI-unl*:
$w \models f\ x \Longrightarrow w \models (\exists\exists\ x.\ f\ x)$
by (*meson exist-state-d-def*)

lemma *EExNoDep*:
$\vdash (\exists\exists\ x.\ g) = g$
proof −
have *1*: $\vdash g \longrightarrow (\exists\exists\ x.\ g)$ **by** (*meson EExI*)
have *2*: $\bigwedge x. \vdash g \longrightarrow g$ **by** *simp*
have *3*: $\vdash (\exists\exists\ x.\ g) \longrightarrow g$ **using** *2* **by** (*meson EExE*)
from *1 3* **show** *?thesis* **using** *int-iffI* **by** *blast*
qed

lemma *AAxNoDep*:
$\vdash (\forall\forall\ x.\ g) = g$
using *EExNoDep*[*of LIFT*($\neg\ g$)] *AAxDef EExE EExI*
by (*simp add*: *exist-state-d-def forall-state-d-def intI*)

lemma *EExEqvRule*:
assumes $\bigwedge x. \vdash f\ x = g\ x$
shows $\vdash (\exists\exists\ x.\ f\ x) = (\exists\exists\ x.\ g\ x)$
by (*metis EExE EExI assms int-iffD1 int-iffD2 int-iffI lift-imp-trans*)

lemma *AAxImpEEx*:
$\vdash (\forall\forall\ x.\ f\ x) \longrightarrow (\exists\exists\ x.\ f\ x)$
by (*simp add*: *exist-state-d-def forall-state-d-def intI*)

lemma *EExImpRule*:
assumes $\vdash f\ x \longrightarrow g\ x$
shows $\vdash (\exists\exists\ x.\ f\ x \longrightarrow\ g\ x)$
using *assms* **by** (*meson MP EExI*)

lemma *EExImpRuleDist*:

assumes $\vdash f\ x \longrightarrow g\ x$
shows $\vdash (\forall\forall\ x.\ f\ x) \longrightarrow (\exists\exists\ x.\ g\ x)$
proof −
have *1*: $\vdash (f\ x) \longrightarrow (\exists\exists\ x.\ g\ x)$ **using** *EExI assms lift-imp-trans* **by** *blast*
have *2*: $\vdash \neg(f\ x) \vee (\exists\exists\ x.\ g\ x)$ **using** *1* **by** *auto*
have *3*: $\vdash \neg(f\ x) \longrightarrow (\exists\exists\ x.\ \neg(f\ x))$ **by** (*meson EExI*)
have *4*: $\vdash (\exists\exists\ x.\ \neg(f\ x)) = (\neg(\forall\forall\ x.\ f\ x))$ **using** *AAxDef* **by** *fastforce*
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *EExImpNoDepDist*:
assumes $\vdash f \longrightarrow g\ x$
shows $\vdash f \longrightarrow (\exists\exists\ x.\ g\ x)$
using *assms* **by** (*metis EExI lift-imp-trans*)

lemma *EExOrDist-1*:
$\vdash (\exists\exists\ x.\ h\ x) \longrightarrow (\exists\exists\ x.\ (f\ x) \vee (h\ x))$
proof −
have *1*: $\bigwedge x.\ \vdash h\ x \longrightarrow f\ x \vee h\ x$ **by** (*simp add*: *Valid-def*)
have *2*: $\bigwedge x.\ \vdash f\ x \vee h\ x \longrightarrow (\exists\exists\ x.\ (f\ x) \vee (h\ x))$ **by** (*meson EExI*)
have *3*: $\bigwedge x.\ \vdash h\ x \longrightarrow (\exists\exists\ x.\ (f\ x) \vee (h\ x))$ **using** *1 2* **by** (*meson lift-imp-trans*)
from *3* **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-2*:
$\vdash (\exists\exists\ x.\ f\ x) \longrightarrow (\exists\exists\ x.\ (f\ x) \vee (h\ x))$
proof −
have *1*: $\bigwedge x.\ \vdash f\ x \longrightarrow f\ x \vee h\ x$ **by** (*simp add*: *Valid-def*)
have *2*: $\bigwedge x.\ \vdash f\ x \vee h\ x \longrightarrow (\exists\exists\ x.\ (f\ x) \vee (h\ x))$ **by** (*meson EExI*)
have *3*: $\bigwedge x.\ \vdash f\ x \longrightarrow (\exists\exists\ x.\ (f\ x) \vee (h\ x))$ **using** *1 2* **by** (*meson lift-imp-trans*)
from *3* **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-3*:
$\vdash (\exists\exists\ x.\ f\ x) \vee (\exists\exists\ x.\ h\ x) \longrightarrow (\exists\exists\ x.\ (f\ x) \vee (h\ x))$
using *EExOrDist-2 EExOrDist-1* **by** *fastforce*

lemma *EExOrDist-4*:
$\vdash (\exists\exists\ x.\ (f\ x) \vee (h\ x)) \longrightarrow (\exists\exists\ x.\ f\ x) \vee (\exists\exists\ x.\ h\ x)$
proof −
have *1*: $\bigwedge x.\ \vdash (f\ x) \vee (h\ x) \longrightarrow (\exists\exists\ x.\ f\ x) \vee (\exists\exists\ x.\ h\ x)$
by (*simp add*: *EExI-unl intI*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExOrDist*:
$\vdash ((\exists\exists\ x.\ f\ x) \vee (\exists\exists\ x.\ h\ x)) = (\exists\exists\ x.\ (f\ x) \vee (h\ x))$
using *EExOrDist-3 EExOrDist-4* **by** *fastforce*

lemma *EExOrImport-1*:

$\vdash g \longrightarrow (\exists\exists\ x.\ g \vee (f\ x))$
by (*simp add*: *EExI-unl Valid-def*)

lemma *EExOrImport-2*:
$\vdash (\exists\exists\ x.\ f\ x) \longrightarrow (\exists\exists\ x.\ g \vee (f\ x))$
by (*simp add*: *EExOrDist-1*)

lemma *EExOrImport-3*:
$\vdash (g \vee (\exists\exists\ x.\ f\ x)) \longrightarrow (\exists\exists\ x.\ g \vee (f\ x))$
using *EExOrImport-1 EExOrImport-2* **by** *fastforce*

lemma *EExOrImport-4*:
$\vdash (\exists\exists\ x.\ g \vee f\ x) \longrightarrow (g \vee (\exists\exists\ x.\ f\ x))$
proof −
have *1*: $\bigwedge x. \vdash g \vee f\ x \longrightarrow g \vee (\exists\exists\ x.\ f\ x)$ **by** (*meson EExI int-iffD2 int-simps*(*27*) *Prop04 Prop08*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExOrImport*:
$\vdash (g \vee (\exists\exists\ x.\ f\ x)) = (\exists\exists\ x.\ g \vee f\ x)$
by (*metis EExOrImport-3 EExOrImport-4 int-iffI*)

lemma *EExAndImport-1*:
$\vdash g \wedge (\exists\exists\ x.\ f\ x) \longrightarrow (\exists\exists\ x.\ g \wedge f\ x)$
proof −
have *1*: $\vdash (g \wedge (\exists\exists\ x.\ f\ x) \longrightarrow (\exists\exists\ x.\ g \wedge f\ x)) = ((\exists\exists\ x.\ f\ x) \longrightarrow (g \longrightarrow (\exists\exists\ x.\ g \wedge f\ x)))$
by *fastforce*
have *2*: $\bigwedge x. \vdash f\ x \longrightarrow (g \longrightarrow (\exists\exists\ x.\ g \wedge f\ x))$
using *EExI*[*of* $\lambda x.\ LIFT(g \wedge f\ x)$] **by** *fastforce*
hence *3*: $\vdash (\exists\exists\ x.\ f\ x) \longrightarrow (g \longrightarrow (\exists\exists\ x.\ g \wedge f\ x))$
by (*simp add*: *EExE*)
from *1 3* **show** *?thesis* **by** *auto*
qed

lemma *EExAndImport-2*:
$\vdash (\exists\exists\ x.\ g \wedge f\ x) \longrightarrow g \wedge (\exists\exists\ x.\ f\ x)$
proof −
have *1*: $\bigwedge x. \vdash g \wedge f\ x \longrightarrow g \wedge (\exists\exists\ x.\ f\ x)$
by (*metis EExI int-iffD2 lift-and-com lift-imp-trans Prop12*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExAndImport*:
$\vdash (g \wedge (\exists\exists\ x.\ f\ x)) = (\exists\exists\ x.\ g \wedge f\ x)$
by (*simp add*: *EExAndImport-1 EExAndImport-2 int-iffI*)

lemma *EExAndDist*:
assumes $\vdash f\ x \wedge g\ x$
shows $\vdash (\exists\exists\ x.\ f\ x) \wedge (\exists\exists\ x.\ g\ x)$
proof −

have *1*: $\vdash f\ x$ **using** *assms* **by** *fastforce*
have *2*: $\vdash g\ x$ **using** *assms* **by** *fastforce*
have *3*: $\vdash (\exists\exists\ x.\ f\ x)$ **using** *1* **by** (*meson EExI MP*)
have *4*: $\vdash (\exists\exists\ x.\ g\ x)$ **using** *2* **by** (*meson EExI MP*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *EExAndNoDepDist*:
assumes $\vdash f \wedge g\ x$
shows $\vdash f \wedge (\exists\exists\ x.\ g\ x)$
proof −
have *1*: $\vdash f$ **using** *assms* **by** *fastforce*
have *2*: $\vdash g\ x$ **using** *assms* **by** *fastforce*
have *3*: $\vdash (\exists\exists\ x.\ g\ x)$ **using** *2* **by** (*meson EExI MP*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *Spec*:
$\vdash (\forall\forall\ x.\ f\ x) \longrightarrow f\ x$
proof −
have *1*: $\vdash \neg(f\ x) \longrightarrow (\exists\exists\ x.\ \neg(f\ x))$ **by** (*meson EExI*)
have *2*: $\vdash \neg(\exists\exists\ x.\ \neg(f\ x)) \longrightarrow f\ x$ **using** *1* **by** *auto*
from *2* **show** *?thesis* **using** *AAxDef* **by** *fastforce*
qed

lemma *AAxE*:
assumes $\vdash (\forall\forall\ x.\ f\ x)$
$\vdash f\ x \longrightarrow g$
shows $\vdash g$
using *MP Spec assms*(*1*) *assms*(*2*) **by** *blast*

lemma *AAxI*:
assumes $\bigwedge x.\ \vdash f\ x$
shows $\vdash (\forall\forall\ x.\ f\ x)$
using *assms* **by** (*simp add*: *Valid-def exist-state-d-def forall-state-d-def*)

lemma *AAxEqvRule*:
assumes $\bigwedge x.\ \vdash f\ x = g\ x$
shows $\vdash (\forall\forall\ x.\ f\ x) = (\forall\forall\ x.\ g\ x)$
unfolding *forall-state-d-def* **using** *assms EExEqvRule*[*of* $\lambda x.\ (LIFT(\neg\ f\ x))$ $\lambda x.\ (LIFT(\neg g\ x))$]
by *fastforce*

lemma *AAxAndDist*:
$\vdash (\forall\forall\ x.\ (f\ x) \wedge (g\ x)) = ((\forall\forall\ x.\ f\ x) \wedge (\forall\forall\ x.\ g\ x))$
proof −
have *1*: $\bigwedge x\ .\ \vdash (\neg(f\ x) \vee \neg(g\ x)) = (\neg((f\ x) \wedge (g\ x)))$
by *auto*
have *2*: $\vdash (\exists\exists\ x.\ \neg(f\ x) \vee \neg(g\ x)) = (\exists\exists\ x.\ \neg((f\ x) \wedge (g\ x)))$
using *1* **by** (*simp add*: *EExEqvRule*)
have *3*: $\vdash(\exists\exists\ fa.\ \neg\ (f\ fa \wedge g\ fa)) = (\exists\exists\ fa.\ \neg\ f\ fa \vee \neg\ g\ fa)$

using *2* **by** *auto*
have *4*: $\vdash(\neg\ (\neg\ (\exists\exists\ fa.\ \neg\ f\ fa)\ \wedge\ \neg\ (\exists\exists\ f.\ \neg\ g\ f))) = ((\exists\exists\ fa.\ \neg\ f\ fa)\ \vee\ (\exists\exists\ f.\ \neg\ g\ f))$
by *auto*
have $\vdash((\exists\exists\ fa.\ \neg\ f\ fa)\ \vee\ (\exists\exists\ f.\ \neg\ g\ f)) = (\exists\exists\ fa.\ \neg\ f\ fa\ \vee\ \neg\ g\ fa)$
by (*simp add*: *EExOrDist inteq-reflection*)
then have *5*: $\vdash(\neg\ (\exists\exists\ fa.\ \neg\ f\ fa)\ \wedge\ \neg\ (\exists\exists\ f.\ \neg\ g\ f)) = (\neg\ (\exists\exists\ fa.\ \neg\ f\ fa\ \vee\ \neg\ g\ fa))$
by *auto*
show *?thesis* **using** *3 5* **unfolding** *forall-state-d-def* **by** *fastforce*
qed

lemma *AAxAndImport*:
$\vdash\ (g\ \wedge\ (\forall\forall\ x.\ f\ x)) = (\forall\forall\ x.\ g\ \wedge\ f\ x)$
proof −
have *1*: $\vdash\ (\neg\ g\ \vee\ (\exists\exists\ x.\ \neg(f\ x))) = (\exists\exists\ x.\ \neg\ g\ \vee\ \neg(f\ x))$
by (*simp add*: *EExOrImport*)
have *2*: $\vdash\ (\ (\exists\exists\ x.\ \neg(f\ x))) = (\neg((\forall\forall\ x.\ f\ x)))$
using *AAxDef* **by** *fastforce*
have *3*: $\vdash\ (\ \neg\ g\ \vee\ (\exists\exists\ x.\ \neg(f\ x))) = (\neg(g\ \wedge\ (\forall\forall\ x.\ f\ x)))$
using *2* **by** *fastforce*
have *4*: $\bigwedge x.\ \vdash\ (\neg\ g\ \vee\ \neg(f\ x)) = (\neg(g\ \wedge\ f\ x))$
by *auto*
have *5*: $\vdash\ (\exists\exists\ x.\ \neg\ g\ \vee\ \neg(f\ x)) = (\exists\exists\ x.\ \neg(g\ \wedge\ f\ x))$
using *4* **by** (*simp add*: *EExEqvRule*)
have *6*: $\vdash\ (\exists\exists\ x.\ \neg(g\ \wedge\ f\ x)) = (\neg(\forall\forall\ x.\ g\ \wedge\ f\ x))$
using *AAxDef* **by** *fastforce*
have *7*: $\vdash\ (\neg(g\ \wedge\ (\forall\forall\ x.\ f\ x))) = (\neg(\forall\forall\ x.\ g\ \wedge\ f\ x))$
by (*metis 1 3 5 6 inteq-reflection*)
from *7* **show** *?thesis* **by** *fastforce*
qed

lemma *AAxOrImport*:
$\vdash\ (g\ \vee\ (\forall\forall\ x.\ f\ x)) = (\forall\forall\ x.\ g\ \vee\ f\ x)$
proof −
have *1*: $\vdash\ (\neg\ g\ \wedge\ (\exists\exists\ x.\ \neg(f\ x))) = (\exists\exists\ x.\ \neg\ g\ \wedge\ \neg(f\ x))$ **by** (*simp add*: *EExAndImport*)
have *2*: $\vdash\ (\exists\exists\ x.\ \neg(f\ x)) =\ (\neg((\forall\forall\ x.\ f\ x)))$ **using** *AAxDef* **by** *fastforce*
have *3*: $\vdash\ (\ \neg\ g\ \wedge\ (\exists\exists\ x.\ \neg(f\ x))) = (\neg(g\ \vee\ (\forall\forall\ x.\ f\ x)))$ **using** *2* **by** *fastforce*
have *4*: $\bigwedge x.\ \vdash\ (\neg\ g\ \wedge\ \neg(f\ x)) = (\neg(g\ \vee\ f\ x))$ **by** *auto*
have *5*: $\vdash\ (\exists\exists\ x.\ \neg\ g\ \wedge\ \neg(f\ x)) = (\exists\exists\ x.\ \neg(g\ \vee\ f\ x))$ **using** *4* **by** (*simp add*: *EExEqvRule*)
have *6*: $\vdash\ (\exists\exists\ x.\ \neg(g\ \vee\ f\ x)) = (\neg(\forall\forall\ x.\ g\ \vee\ f\ x))$ **using** *AAxDef* **by** *fastforce*
have *7*: $\vdash\ (\neg(g\ \vee\ (\forall\forall\ x.\ f\ x))) = (\neg(\forall\forall\ x.\ g\ \vee\ f\ x))$ **by** (*metis 1 3 5 6 inteq-reflection*)
from *7* **show** *?thesis* **by** *auto*
qed

lemma *EExImpChopRule*:
assumes $\vdash\ f\ x\ \longrightarrow\ g\ x$
shows $\vdash\ (\exists\exists\ x.\ h;(f\ x)\ \longrightarrow\ h;(g\ x))$
using *RightChopImpChop*[*of f x g x h*]
EExImpRule[*of* $\lambda x.\ LIFT(h;(f\ x))\ x\ \lambda x.\ LIFT(h;(g\ x))$] *assms* **by** *auto*

lemma *EExChopRight*:
$\vdash (\exists\exists\ x.\ (f\ x);g) \longrightarrow (\exists\exists\ x.\ f\ x);g$
proof −
have *1*: $\bigwedge x. \vdash (f\ x);g \longrightarrow (\exists\exists\ x.\ f\ x);g$ **by** (*simp add*: *EExI LeftChopImpChop*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExChopRightNoDep*:
$\vdash (\exists\exists\ x.\ (f\ x);g) = (\exists\exists\ x.\ (f\ x));g$
by (*auto simp add*: *exist-state-d-def Valid-def itl-defs*)

lemma *EExChopLeft* :
$\vdash (\exists\exists\ x.\ g;(f\ x)\ ) \longrightarrow g;(\exists\exists\ x.\ f\ x)$
proof −
have *1*: $\bigwedge x. \vdash g;(f\ x) \longrightarrow g;(\exists\exists\ x.\ f\ x)$ **by** (*simp add*: *EExI RightChopImpChop*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExChopLeftNoDep*:
$\vdash (\exists\exists\ x.\ g;(f\ x)\ ) = g;(\exists\exists\ x.\ f\ x)$
by (*auto simp add*: *exist-state-d-def Valid-def itl-defs*)

lemma *EExEExChopEqvEExEExChop*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (f\ v);(g\ y)\ )) = (\exists\exists\ y.\ (\exists\exists\ v.\ (f\ v);(g\ y)\ ))$
by (*simp add*: *exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExEExChopEqvEExChopEExA*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (f\ v);(g\ y)\ )) = (\exists\exists\ v.\ (f\ v);(\exists\exists\ y.\ (g\ y)\ )\ )$
by (*simp add*: *exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExEExChopEqvEExChopEExB*:
$\vdash (\exists\exists\ y.\ (\exists\exists\ v.\ (f\ v);(g\ y)\ )) = (\exists\exists\ y.\ (\exists\exists\ v.\ (f\ v)); (g\ y))$
by (*simp add*: *exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExEExChopEqvEExChopEExC*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (f\ v);(g\ y)\ )) = (\exists\exists\ v.\ (f\ v));(\exists\exists\ y.\ (g\ y))$
by (*simp add*: *exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExChopDistA*:
$\vdash (\exists\exists\ v.\ (f\ v);(g\ v)) \longrightarrow (\ \exists\exists\ v.\ (f\ v))\ ;\ (\exists\exists\ v.\ (g\ v))$
by (*simp add*: *exist-state-d-def Valid-def itl-defs*)
blast

lemma *EExChopDistB*:
$\vdash (\ \exists\exists\ v.\ (f\ v))\ schop\ (skip\ schop\ (\exists\exists\ v.\ (g\ v))) \longrightarrow$
$(\exists\exists\ v\ v1.\ (f\ v)\ schop\ (skip\ schop\ (g\ v1)))$
by (*auto simp add*: *exist-state-d-def Valid-def itl-defs*)

lemma *EExChopDistC*:
$\vdash (\ \exists\exists\ v.\ (f\ v))\ schop\ (\ (\exists\exists\ v.\ (g\ v))) \longrightarrow$

$(\exists\exists\ v\ v1.\ (f\ v)\ schop\ (\ (g\ v1)))$
by (*auto simp add*: *exist-state-d-def Valid-def itl-defs*)

lemma *ExLenOrInf*:
$\vdash (\exists\ n.\ len(n)) \vee inf$
using *nfinite-nlength-enat* **by** (*auto simp add*: *Valid-def len-defs itl-defs*)

lemma *CSPowerChop*:
$\vdash (f^\star) = (\exists n.\ fpower\ (f \wedge more)\ n);(empty \vee (f \wedge more) \wedge inf)$
by (*simp add*: *chopstar-d-def fpowerstar-d-def powerstar-d-def Valid-def*)

lemma *ExChopRightNoDep*:
$\vdash (\exists\ x.\ (f\ x);g) = (\exists\ x.\ (f\ x));g$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *ExChopLeftNoDep*:
$\vdash (\exists\ x.\ g;(f\ x)\ ) = g;(\exists\ x.\ f\ x)$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *ExExEqvExEx*:
$\vdash (\exists\ x.\ (\exists\ y.\ (f\ x);(g\ y))) = (\exists\ y.\ (\exists\ x.\ (f\ x);(g\ y)))$
by (*auto simp add*: *Valid-def itl-defs*)

3.11 Bind operator

lemma *Ex-box-eq-sfxbnd*:
$\vdash (\exists\exists v.\ \Box(\$v = f1) \wedge f2\ v) = (sfxbnd\ v\ as\ f1 : f2\ v)$
by (*simp add*: *itl-defs Valid-def sfxbnd-defs exist-state-d-def ndropn-nfirst*)

lemma *Ex-bf-eq-pfxbnd*:
$\vdash (\exists\exists v.\ bf(!v = f1) \wedge f2\ v) = (pfxbnd\ v\ as\ f1 : f2\ v)$
by (*simp add*: *itl-defs Valid-def pfxbnd-defs exist-state-d-def ntaken-nlast*)

lemma *sfxbnd-true*:
$\vdash (sfxbnd\ v\ as\ \#True : f2\ v) = (\exists\exists\ v.\ (\Box\ \$v) \wedge f2\ v)$
by (*simp add*: *itl-defs Valid-def sfxbnd-defs exist-state-d-def ndropn-nfirst*)

end

3.12 Time reversal operator

theory *TimeReversal*
imports
SChopTheorems FOTheorems Omega
begin

Time reversal operator is defined in [12]. Note: in this theory only finite intervals can be reversed.

3.12.1 Definition

definition *freverse-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *freverse-d F* ≡ λ *s*. *nfinite s* ∧ (*nrev s* ⊨ *F*)

definition *ereverse-d* :: (*'a*::*world*, *'b*) *formfun* ⇒ (*'a*, *'b*) *formfun*
where *ereverse-d f* ≡ λ *s*. (*if nfinite s then* (*nrev s* ⊨ *f*) *else* (ϵ (*x*::*'b*). *x*=*x*))

syntax
-freverse-d :: *lift* ⇒ *lift* ((-^{r}) [*85*] *85*)
-ereverse-d :: *lift* ⇒ *lift* ((-^{R}) [*100*] *99*)

syntax (*ASCII*)
-freverse-d :: *lift* ⇒ *lift* ((*frev* -) [*85*] *85*)
-ereverse-d :: *lift* ⇒ *lift* ((*erev* -) [*100*] *99*)

translations
-freverse-d ⇌ *CONST freverse-d*
-ereverse-d ⇌ *CONST ereverse-d*

definition *wfreverse-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *wfreverse-d F* ≡ *LIFT* (¬((¬*F*)^{r}))

syntax
-wfreverse-d :: *lift* ⇒ *lift* ((*wrev* -) [*85*] *85*)

syntax (*ASCII*)
-wfreverse-d :: *lift* ⇒ *lift* ((*wrev* -) [*85*] *85*)

translations
-wfreverse-d ⇌ *CONST wfreverse-d*

3.12.2 Finite theorems

lemma *FiniteChopEqv*:
assumes ⊢ *finite* ⟶ *f* = *g*
shows ⊢ *finite* ⟶ *f*;*h* = *g*;*h*
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteTransEqv*:
assumes ⊢ *finite* ⟶ *f* = *g*
⊢ *finite* ⟶ *g* = *h*
shows ⊢ *finite* ⟶ *f* = *h*
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteOrEqv*:
assumes ⊢ *finite* ⟶ *f1* = *f2*
⊢ *finite* ⟶ *g1* = *g2*
shows ⊢ *finite* ⟶ (*f1* ∨ *g1*) = (*f2* ∨ *g2*)

using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteAndEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *f1* = *f2*
$\vdash$ *finite* $\longrightarrow$ *g1* = *g2*
shows $\vdash$ *finite* $\longrightarrow$ (*f1* $\wedge$ *g1*) = (*f2* $\wedge$ *g2*)
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteNotEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *f1* = *f2*
shows $\vdash$ *finite* $\longrightarrow$ ($\neg$ *f1*) = ($\neg$ *f2*)
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteWNextEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* =*g*
shows $\vdash$ *finite* $\longrightarrow$ *wnext f* = *wnext g*
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteWPrevEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* =*g*
shows $\vdash$ *finite* $\longrightarrow$ *wprev f* = *wprev g*
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteInitonlyEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* = *g*
shows $\vdash$ *finite* $\longrightarrow$ *initonly f* = *initonly g*
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteKeepEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* = *g*
shows $\vdash$ *finite* $\longrightarrow$ *keep f* = *keep g*
using *assms*
by (*auto simp add*: *Valid-def keep-d-def itl-defs nsubn-nfinite*)

lemma *FiniteHaltEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* = *g*
shows $\vdash$ *finite* $\longrightarrow$ *halt f* = *halt g*
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteDaAlt*:
$\vdash$ *finite* $\longrightarrow$ *da f* = (#*True*;*f*);#*True*
unfolding *da-d-def*
by (*metis ChopAssoc FiniteChopEqv int-simps*(*13*) *int-simps*(*9*) *inteq-reflection*)

lemma *ChopCsImpCSChop*:
$\vdash$ (*f* $\wedge$ *finite*);(*schopstar f*) $\longrightarrow$ (*schopstar f*);(*f* $\wedge$ *finite*)
by (*metis SChopstar-WPowerstar WPowerstar-slide-var int-iffD2 inteq-reflection*)

lemma *CSChopImpChopCS*:
$\vdash$ (*schopstar f*);(*f* $\wedge$ *finite*) $\longrightarrow$ (*f* $\wedge$ *finite*);(*schopstar f*)
by (*metis SChopstar-WPowerstar WPowerstar-slide-var int-iffD1 inteq-reflection*)

lemma *NextDiamondEqvDiamondNext*:
$\vdash$ $\bigcirc$($\Diamond$ *f*) = $\Diamond$($\bigcirc$*f*)
proof −
have *1*: $\vdash$ *finite*;*skip* = *skip*;*finite*
by (*simp add*: *FiniteChopSkipEqvSkipChopFinite*)
hence *2*: $\vdash$ (*finite*;*skip*);*f* =(*skip*;*finite*);*f* **using** *LeftChopEqvChop* **by** *blast*
have *3*: $\vdash$ (*finite*;*skip*);*f* = *finite*;(*skip*;*f*)
by (*meson ChopAssoc ChopEmpty Prop04*)
have *4*: $\vdash$ (*skip*;*finite*);*f* = *skip*;(*finite*;*f*)
by (*meson ChopAssoc Prop11*)
from *2 3 4* **show** *?thesis* **by** (*metis int-eq next-d-def sometimes-d-def*)
qed

lemma *WeakNextBoxInduct*:
assumes $\vdash$ *wnext* ($\Box$ *f*) $\longrightarrow$ *f*
shows $\vdash$ *finite* $\longrightarrow$ *f*
proof −
have *1*: $\vdash$ *wnext* ($\Box$ *f*) $\longrightarrow$ *f* **using** *assms* **by** *blast*
hence *2*: $\vdash$ $\neg$ *f* $\longrightarrow$ $\neg$ (*wnext* ($\Box$ *f*)) **by** *fastforce*
hence *3*: $\vdash$ $\neg$ *f* $\longrightarrow$ $\bigcirc$ ($\neg$ ($\Box$ *f*)) **by** (*simp add*: *wnext-d-def*)
have *4*: $\vdash$ ($\neg$ ($\Box$ *f*)) = ($\Diamond$ ($\neg$ *f*)) **by** (*auto simp*: *always-d-def*)
hence *5*: $\vdash$ $\bigcirc$($\neg$ ($\Box$ *f*)) = $\bigcirc$ ($\Diamond$ ($\neg$ *f*)) **using** *NextEqvNext* **by** *blast*
have *6*: $\vdash$ $\neg$ *f* $\longrightarrow$ $\bigcirc$ ($\Diamond$ ($\neg$ *f*)) **using** *3 5* **by** *fastforce*
have *7*: $\vdash$ $\bigcirc$ ($\Diamond$ ($\neg$ *f*)) = $\Diamond$($\bigcirc$ ($\neg$ *f*)) **using** *NextDiamondEqvDiamondNext* **by** *blast*
have *8*: $\vdash$ $\neg$ *f* $\longrightarrow$ $\Diamond$($\bigcirc$ ($\neg$ *f*)) **using** *6 7* **by** *fastforce*
have *9*: $\vdash$ $\Diamond$($\neg$ *f*) $\longrightarrow$ $\Diamond$($\Diamond$($\bigcirc$ ($\neg$ *f*))) **using** *8 DiamondImpDiamond* **by** *blast*
have *10*: $\vdash$ $\Diamond$($\Diamond$($\bigcirc$ ($\neg$ *f*))) = $\Diamond$($\bigcirc$ ($\neg$ *f*)) **using** *DiamondDiamondEqvDiamond* **by** *blast*
have *11*: $\vdash$ $\Diamond$($\neg$ *f*) $\longrightarrow$ $\Diamond$($\bigcirc$ ($\neg$ *f*)) **using** *9 10* **by** *fastforce*
have *12*: $\vdash$ $\Diamond$($\neg$ *f*) $\longrightarrow$ $\bigcirc$ ($\Diamond$ ($\neg$ *f*)) **using** *7 11* **by** *fastforce*
hence *13*: $\vdash$ *finite* $\longrightarrow$ $\neg$($\Diamond$($\neg$ *f*)) **using** *NextLoop* **by** *blast*
hence *14*: $\vdash$ *finite* $\longrightarrow$ $\Box$*f* **by** (*simp add*: *always-d-def*)
have *15*: $\vdash$ $\Box$*f* $\longrightarrow$ *f* **using** *BoxElim* **by** *blast*
from *14 15* **show** *?thesis* **by** (*metis lift-imp-trans*)
qed

3.12.3 Time reversal Rules

lemma *WRevRev*:
$\vdash$ (*wrev f*) = (*finite* $\longrightarrow$ f^r)
by (*simp add*: *Valid-def wfreverse-d-def freverse-d-def finite-defs nfinite-nrev*)

lemma *RevWRev*:
$\vdash$ (*frev f*) = (*finite* $\wedge$ *wrev f*)
by (*auto simp add*: *Valid-def wfreverse-d-def freverse-d-def finite-defs nfinite-nrev*)

lemma *RevWRevAlt*:
$\vdash$ (*frev f*) = ($\neg$((*wrev* ($\neg$*f*))))
unfolding *wfreverse-d-def*
by *simp*

lemma *EExRev* :
$\vdash$ ($\exists\exists$ *x*. *F x*)^r = ($\exists\exists$ *x*. (*F x*)^r)
by (*simp add*: *Valid-def exist-state-d-def freverse-d-def*)

lemma *EExWRev* :
$\vdash$ (*wrev* ($\exists\exists$ *x*. *F x*)) = ($\exists\exists$ *x*. (*wrev* (*F x*)))
by (*simp add*: *Valid-def exist-state-d-def wfreverse-d-def freverse-d-def*)

lemma *rev-const* :
$\vdash$ *finite* $\longrightarrow$ (#*c*)^R = #*c*
by (*auto simp add*: *Valid-def itl-defs ereverse-d-def*)

lemma *rev-fun1* :
$\vdash$ *finite* $\longrightarrow$ (*f*<*x*>)^r = *f*<x^r>
by (*auto simp*: *itl-defs freverse-d-def*)

lemma *rev-fun1-alt* :
$\vdash$ (*f*<*x*>)^r = (*f*<x^r> $\wedge$ *finite*)
by (*auto simp*: *itl-defs freverse-d-def*)

lemma *wrev-fun1* :
$\vdash$ *finite* $\longrightarrow$(*wrev* (*f*<*x*>)) = *f*<(*wrev x*)>
by (*auto simp add*: *itl-defs wfreverse-d-def freverse-d-def*)

lemma *wrev-fun1-alt* :
$\vdash$ (*wrev* (*f*<*x*>)) = (*finite* $\longrightarrow$ *f*<(*wrev x*)>)
by (*auto simp add*: *itl-defs wfreverse-d-def freverse-d-def*)

lemma *rev-fun1-e* :
$\vdash$ *finite* $\longrightarrow$ (*f*<*x*>)^R = *f*<x^R>
by (*auto simp*: *itl-defs ereverse-d-def*)

lemma *rev-fun2*:
$\vdash$ *finite* $\longrightarrow$ (*f*<*x*,*y*>)^r = *f*<x^r,y^r>
by (*auto simp*: *itl-defs freverse-d-def*)

lemma *rev-fun2-alt*:
$\vdash\ (f{<}x,y{>})^r = (f{<}x^r,y^r{>} \wedge \mathit{finite})$
by (*auto simp*: *itl-defs freverse-d-def*)

lemma *wrev-fun2*:
$\vdash \mathit{finite} \longrightarrow (\mathit{wrev}\ (f{<}x,y{>})) = f{<}(\mathit{wrev}\ x),(\mathit{wrev}\ y){>}$
by (*auto simp add*: *itl-defs wfreverse-d-def freverse-d-def*)

lemma *wrev-fun2-alt* :
$\vdash\ (\mathit{wrev}\ (f{<}x,y{>})) = (\mathit{finite} \longrightarrow f{<}(\mathit{wrev}\ x),\ (\mathit{wrev}\ y){>}\ )$
by (*auto simp add*: *itl-defs wfreverse-d-def freverse-d-def*)

lemma *rev-fun2-e*:
$\vdash \mathit{finite} \longrightarrow (f{<}x,y{>})^R = f{<}x^R,y^R{>}$
by (*auto simp*: *itl-defs ereverse-d-def*)

lemma *rev-fun3*:
$\vdash \mathit{finite} \longrightarrow (f{<}x,y,z{>})^r = f{<}x^r,y^r,z^r{>}$
by (*auto simp*: *itl-defs freverse-d-def*)

lemma *rev-fun3-alt*:
$\vdash\ (f{<}x,y,z{>})^r = (f{<}x^r,y^r,z^r{>} \wedge \mathit{finite})$
by (*auto simp*: *itl-defs freverse-d-def*)

lemma *wrev-fun3*:
$\vdash \mathit{finite} \longrightarrow (\mathit{wrev}\ (f{<}x,y,z{>})) = f{<}(\mathit{wrev}\ x),(\mathit{wrev}\ y),(\mathit{wrev}\ z){>}$
by (*auto simp add*: *itl-defs wfreverse-d-def freverse-d-def*)

lemma *wrev-fun3-alt* :
$\vdash\ (\mathit{wrev}\ (f{<}x,y,z{>})) = (\mathit{finite} \longrightarrow f{<}(\mathit{wrev}\ x),\ (\mathit{wrev}\ y),\ (\mathit{wrev}\ z){>}\ )$
by (*auto simp add*: *itl-defs wfreverse-d-def freverse-d-def*)

lemma *rev-fun3-e*:
$\vdash \mathit{finite} \longrightarrow (f{<}x,y,z{>})^R = f{<}x^R,y^R,z^R{>}$
by (*auto simp*: *itl-defs ereverse-d-def*)

lemma *rev-forall*:
$\vdash (\forall\ x.\ P\ x)^r = (\forall\ x.\ (P\ x)^r)$
by (*auto simp*: *freverse-d-def*)

lemma *wrev-forall*:
$\vdash (\mathit{wrev}\ (\forall\ x.\ P\ x)) = (\forall\ x.\ (\mathit{wrev}\ (P\ x)))$
by (*auto simp*: *wfreverse-d-def freverse-d-def*)

lemma *rev-exists*:
$\vdash (\exists\ x.\ P\ x)^r = (\exists\ x.\ (P\ x)^r)$
by (*auto simp*: *freverse-d-def*)

lemma *wrev-exists*:

$\vdash$ (*wrev* (∃ *x*. *P x*)) = (∃ *x*. (*wrev* (*P x*)))
by (*auto simp*: *wfreverse-d-def freverse-d-def*)

lemma *rev-exists1*:
$\vdash$ (∃! *x*. *P x*)^r = (∃! *x*. (*P x*)^r)
by (*auto simp*: *freverse-d-def*)

lemma *wrev-exists1*:
$\vdash$ *wrev* (∃! *x*. *P x*) = (*finite* ⟶ (∃! *x*. *wrev* (*P x*)))
proof −
have *1*: $\vdash$ *wrev* (∃! *x*. *P x*) = (*finite* ⟶ *frev* (∃! *x*. *P x*))
using *WRevRev* **by** *blast*
have *2*: $\vdash$ (*finite* ⟶ *frev* (∃! *x*. *P x*)) = (*finite* ⟶ (∃! *x*. *frev* (*P x*)))
by (*metis int-simps*(*27*) *inteq-reflection rev-exists1*)
have *3*: ⋀ *x*. $\vdash$ *frev* (*P x*) = (*wrev* (*P x*) ∧ *finite*)
by (*metis RevWRev inteq-reflection lift-and-com*)
have *4*: $\vdash$ ((∃! *x*. *frev* (*P x*))) = (∃! *x*. *wrev* (*P x*) ∧ *finite*)
using *3* **by** (*simp add*: *Valid-def freverse-d-def itl-defs*)
have *5*: $\vdash$ (∃! *x*. *wrev* (*P x*) ∧ *finite*) = ((∃! *x*. *wrev* (*P x*)) ∧ *finite*)
by *auto*
have *6*: $\vdash$ (*finite* ⟶ ((∃! *x*. *wrev* (*P x*)) ∧ *finite*)) = (*finite* ⟶ (∃! *x*. *wrev* (*P x*)))
by *force*
show *?thesis*
by (*metis 1 2 4 5 6 inteq-reflection*)
qed

lemma *rev-current*:
$\vdash$ *finite* ⟶ (*$v*)^R = (*!v*)
by (*auto simp*: *nnth-nrev itl-def ereverse-d-def nfirst-nrev*)

lemma *rev-next*:
$\vdash$ *finite* ⟶ (*v$*)^R = (*v!*)
by (*auto simp*: *itl-defs ereverse-d-def nfinite-nrev*)
(*metis One-nat-def enat-ord-simps*(*2*) *epred-conv-minus idiff-enat-enat less-Suc0 linorder-not-less nfinite-nlength-enat nnth-nrev one-enat-def the-enat.simps*)

lemma *rev-penult*:
$\vdash$ *finite* ⟶ (*v!*)^R = (*v$*)
by (*auto simp*: *itl-defs ereverse-d-def nfinite-nrev*)
(*metis One-nat-def Suc-ile-eq epred-enat i0-less ndropn-nfirst nfinite-ndropn nfinite-nlength-enat nlast-nrev nrev-ndropn ntaken-nlast the-enat.simps zero-enat-def*)

lemma *rev-fin*:
$\vdash$ *finite* ⟶ (*!v*)^R = (*$v*)
by (*auto simp*: *itl-defs ereverse-d-def nfinite-nrev nlast-nrev*)

lemma *EqvReverseReverse*:

$\vdash$ *finite* $\longrightarrow (f^r)^r = f$
by (*simp add*: *Valid-def itl-defs freverse-d-def nfinite-nrev nrev-nrev*)

lemma *EqvWReverseWReverse*:
$\vdash$ *finite* $\longrightarrow$ (*wrev* (*wrev f*)) = *f*
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-nrev nrev-nrev*)

lemma *EqvReverseReverseAlt*:
$\vdash \ (f^r)^r = (f \wedge$ *finite*)
by (*auto simp add*: *Valid-def itl-defs freverse-d-def nfinite-nrev nrev-nrev*)

lemma *EqvWReverseWReverseAlt*:
$\vdash$ (*wrev* (*wrev f*)) = (*finite* $\longrightarrow$ *f*)
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-nrev nrev-nrev*)

lemma *ReverseEqv*:
($\vdash$ *finite* $\longrightarrow$ *f*) $\longleftrightarrow$ ($\vdash$ *finite* $\longrightarrow f^r$)
by (*simp add*: *Valid-def itl-defs freverse-d-def*)
(*metis nfinite-nrev nrev-nrev*)

lemma *WReverseEqv*:
($\vdash$ *finite* $\longrightarrow$ *f*) $\longleftrightarrow$ ($\vdash$ *finite* $\longrightarrow$ *wrev f*)
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def*)
(*metis nfinite-nrev nrev-nrev*)

lemma *ReverseEqvAlt*:
($\vdash \ f \wedge$ *finite*) $\longleftrightarrow$ ($\vdash f^r$)
by (*auto simp add*: *Valid-def itl-defs freverse-d-def*)

lemma *WReverseEqvAlt*:
($\vdash$ *finite* $\longrightarrow$ *f*) $\longleftrightarrow$ ($\vdash$ *wrev f*)
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def*)
(*metis nfinite-nrev nrev-nrev*)

lemma *RevSkip*:
$\vdash$ *skip*^{r} = *skip*
using *nlength-eq-enat-nfiniteD* **by** (*simp add*: *Valid-def freverse-d-def itl-defs*, *blast*)

lemma *WRevSkip*:
$\vdash$ (*wrev skip*) = (*inf* $\vee$ *skip*)
by (*auto simp add*: *Valid-def wfreverse-d-def freverse-d-def itl-defs*)

lemma *RevFinite*:
$\vdash$ *finite*^{r} = *finite*
by (*simp add*: *Valid-def freverse-d-def finite-defs nfinite-nrev*)

lemma *WRevFinite*:
$\vdash$ (*wrev finite*)
by (*auto simp add*: *Valid-def wfreverse-d-def freverse-d-def itl-defs nfinite-nrev*)

lemma *NotRevInfinite*:
$\vdash \neg$ (inf^r)
by (*simp add*: *Valid-def freverse-d-def infinite-defs nfinite-nrev*)

lemma *WRevInfinite*:
$\vdash$ (*wrev inf*) = *inf*
by (*simp add*: *Valid-def wfreverse-d-def freverse-d-def infinite-defs nfinite-nrev*)

lemma *RevChop*:
$\vdash$ *finite* $\longrightarrow$ $(f;g)^r = (g^r;f^r)$
by (*simp add*: *Valid-def itl-defs freverse-d-def nfinite-nrev*)
(*metis diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *nfinite-nlength-enat nrev-ndropn*
nrev-ntaken the-enat.simps)

lemma *RevYields*:
$\vdash$ *finite* $\longrightarrow$ $(f$ *yields* $g)^r = ((g^r)$ *revyields* $f^r)$
unfolding *yields-d-def revyields-d-def*
by (*meson FiniteChopEqv FiniteNotEqv FiniteTransEqv RevChop rev-fun1*)

lemma *WRevChop*:
$\vdash$ *finite* $\longrightarrow$ (*wrev* $(f;g)$) = ((*wrev* g);(*wrev* f))
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-nrev*)
(*metis diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *nfinite-nlength-enat nrev-ndropn*
nrev-ntaken the-enat.simps)

lemma *RevChopAlt*:
$\vdash$ $(f;g)^r = (g^r;f^r)$
by (*simp add*: *Valid-def itl-defs freverse-d-def nfinite-nrev*)
(*metis diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *nfinite-nlength-enat nrev-ndropn*
nrev-ntaken the-enat.simps)

lemma *RevSChopAlt*:
$\vdash$ $(f \frown g)^r = (g^r \frown f^r)$
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt RevChopAlt int-simps*(*27*)
inteq-reflection rev-fun2-alt schop-d-def)

lemma *WRevChopAlt*:
$\vdash$ (*wrev* $(f;g)$) = ((*wrev* g);(*wrev* f))
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-nrev*)
(*metis diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *nfinite-nlength-enat nrev-ndropn*
nrev-ntaken the-enat.simps)

lemma *WRevSChopAlt*:
$\vdash$ (*wrev* $(f \frown g)$) = ((*wrev* g);(*wrev* f))
unfolding *schop-d-def*
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt EqvWReverseWReverse FiniteImpAnd*
WRevChopAlt int-simps(*4*) *inteq-reflection rev-fun1-alt wfreverse-d-def*)

lemma *RAnd*:
$\vdash (f \wedge g)^r = (f^r \wedge g^r)$
by (*auto simp add*: *Valid-def freverse-d-def*)

lemma *WRAnd*:
$\vdash (wrev\ (f \wedge g)) = (wrev\ f \wedge wrev\ g)$
by (*auto simp add*: *Valid-def wfreverse-d-def freverse-d-def*)

lemma *RevAndFinite*:
$\vdash (f^r \wedge finite) = (f \wedge finite)^r$
by (*metis RAnd RevFinite inteq-reflection*)

lemma *RevAndFiniteAlt*:
$\vdash (f^r \wedge finite) = (f)^r$
by (*metis int-simps*(*20*) *inteq-reflection rev-fun2-alt*)

lemma *WRevAndFinite*:
$\vdash (wrev\ f) \ = (wrev\ (f \wedge finite))$
proof −
have *1*: $\vdash (wrev\ (f \wedge finite)) = ((wrev\ f) \wedge wrev\ finite)$
by (*simp add*: *WRAnd*)
have *2*: $\vdash ((wrev\ f) \wedge wrev\ finite) = (wrev\ f)$
using *RevFinite WRevRev* **by** *fastforce*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *WRevAndFiniteAlt*:
$\vdash ((wrev\ f) \wedge finite) = (frev\ f)$
by (*metis int-simps*(*4*) *inteq-reflection rev-fun1-alt wfreverse-d-def*)

lemma *WRevFiniteImp*:
$\vdash (finite \longrightarrow wrev\ f) = wrev\ f$
by (*metis FiniteImp RevFinite WRevAndFiniteAlt WRevRev inteq-reflection*)

lemma *RMoreEqvMore*:
$\vdash finite \longrightarrow more^r = more$
by (*simp add*: *Valid-def itl-defs freverse-d-def*)

lemma *WRMoreEqvMore*:
$\vdash finite \longrightarrow wrev\ more = more$
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def*)

lemma *RMoreEqvMoreAlt*:
$\vdash \ more^r = (more \wedge finite)$
by (*auto simp add*: *Valid-def itl-defs freverse-d-def*)

lemma *WRMoreEqvMoreAlt*:

$\vdash$ *wrev more* = *more*
by (*auto simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-conv-nlength-enat zero-enat-def*)

lemma *REmptyEqvEmpty*:
$\vdash$ *empty*^{r} = *empty*
by (*auto simp add*: *Valid-def itl-defs freverse-d-def nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *WREmptyEqvEmpty*:
$\vdash$ *wrev empty* = (*finite* $\longrightarrow$ *empty*)
by (*auto simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *RImpRule*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ (*f*^{r}) $\longrightarrow$ (*g*^{r})
using *assms*
by (*auto simp add*: *Valid-def itl-defs freverse-d-def*)

lemma *WRImpRule*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ (*wrev f*) $\longrightarrow$ (*wrev g*)
using *assms*
by (*auto simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def*)

lemma *REqvRule*:
assumes $\vdash$ *f* = *g*
shows $\vdash$ (*f*^{r}) = (*g*^{r})
using *assms*
using *inteq-reflection* **by** *force*

lemma *WREqvRule*:
assumes $\vdash$ *f* = *g*
shows $\vdash$ (*wrev f*) = (*wrev g*)
using *assms*
by (*simp add*: *Prop11 WRImpRule*)

lemma *FPowerFinite*:
$\vdash$ (*fpower* (*f* $\wedge$ *more*) *n*) = ((*fpower* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*)
proof (*induct n*)
case *0*
then show *?case*
by (*metis EmptyImpFinite Prop10 fpower-d-def wpow-0*)
next
case (*Suc n*)
then show *?case*

proof −
have *1*: ⊢ *fpower* (*f* ∧ *more*) (*Suc n*) = (*f* ∧ *fmore*);(*fpower* (*f* ∧ *more*) *n*)
by (*metis AndMoreSChopEqvAndFmoreChop fpower-d-def schop-d-def wpow-Suc*)
have *2*: ⊢ (*f* ∧ *fmore*);(*fpower* (*f* ∧ *more*) *n*) =
((*f* ∧ *fmore*);(*fpower* (*f* ∧ *more*) *n*) ∧ *finite*)
by (*metis ChopImpChop FiniteChopFiniteEqvFinite Prop01 Prop05 Prop10 Suc fmore-d-def int-eq int-iffD1*)
show *?thesis*
by (*metis 1 2 inteq-reflection*)
qed
qed

lemma *RevFPowerChop*:
⊢ (*fpower* (*f* ∧ *more*) *n*)^r = (*fpower* ((*f* ∧ *more*)^r) *n*)
proof
(*induct n*)
case *0*
then show *?case* **using** *REmptyEqvEmpty*
by (*metis fpower-d-def wpow-0*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: ⊢ (*fpower* (*f* ∧ *more*) (*Suc n*))^r = ((*f* ∧ *fmore*);(*fpower* (*f* ∧ *more*) *n*))^r
by (*metis AndMoreSChopEqvAndFmoreChop ChopEmpty fpower-d-def inteq-reflection schop-d-def wpow-Suc*)
have *2*: ⊢ ((*f* ∧ *fmore*);(*fpower* (*f* ∧ *more*) *n*))^r = ((*fpower* (*f* ∧ *more*) *n*);(*f* ∧ *fmore*))^r
by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore FPowerCommute REqvRule inteq-reflection*)
have *3*: ⊢ (*fpower* (*f* ∧ *more*) *n*) = ((*fpower* (*f* ∧ *more*) *n*) ∧ *finite*)
by (*simp add*: *FPowerFinite*)
have *4*: ⊢ ((*fpower* (*f* ∧ *more*) *n*);(*f* ∧ *fmore*))^r = ((*f* ∧ *fmore*)^r;(*fpower* (*f* ∧ *more*) *n*)^r)
by (*metis RevChopAlt*)
show *?thesis*
by (*metis 2 4 AndMoreAndFiniteEqvAndFmore RevAndFinite Suc fpower-d-def inteq-reflection wpow-Suc*)
qed
qed

lemma *FPowerWRevFiniteAbsorb*:
⊢ (*fpower* (*finite* ∧ *wrev* (*f* ∧ *more*)) *n*) = (*fpower* (*wrev* (*f* ∧ *more*)) *n*)
proof (*induct n*)
case *0*
then show *?case*
by (*metis REmptyEqvEmpty fpower-d-def inteq-reflection wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt EqvWReverseWReverseAlt RevWRev WRevAndFinite*

WRevRev fpower-d-def int-eq int-simps(27))
qed

lemma *WRevFPowerChop*:
$\vdash$ *wrev* (*fpower* (*f* $\wedge$ *more*) *n*) = (*finite* $\longrightarrow$ *fpower* (*wrev* (*f* $\wedge$ *more*)) *n*)
proof −
have *1*: $\vdash$ *wrev* (*fpower* (*f* $\wedge$ *more*) *n*) = (*finite* $\longrightarrow$ *frev* (*fpower* (*f* $\wedge$ *more*) *n*))
by (*simp add*: *WRevRev*)
have *2*: $\vdash$ (*finite* $\longrightarrow$ *frev* (*fpower* (*f* $\wedge$ *more*) *n*)) =
(*finite* $\longrightarrow$ (*fpower* ((*f* $\wedge$ *more*)^r) *n*))
using *RevFPowerChop* **by** *fastforce*
have *3*: $\vdash$ (*fpower* ((*f* $\wedge$ *more*)^r) *n*) = (*fpower* (*finite* $\wedge$ *wrev* (*f* $\wedge$ *more*)) *n*)
by (*metis RevFPowerChop RevWRev inteq-reflection*)
have *4*: $\vdash$ (*fpower* (*finite* $\wedge$ *wrev* (*f* $\wedge$ *more*)) *n*) = (*fpower* (*wrev* (*f* $\wedge$ *more*)) *n*)
by (*simp add*: *FPowerWRevFiniteAbsorb*)
show *?thesis*
using *1 2 3 4* **by** *fastforce*
qed

lemma *RevChopstar*:
$\vdash$ (*schopstar f*)^r = (*schopstar* (*f*^{r}))
proof −
have *1*: $\vdash$ (*schopstar f*) = (∃ *n*. *fpower* (*f* $\wedge$ *more*) *n*)
by (*simp add*: *FPowerstardef schopstar-d-def*)
have *2*: $\vdash$ (*schopstar f*)^r = (∃ *n*. *fpower* (*f* $\wedge$ *more*) *n*)^r
using *REqvRule 1* **by** *blast*
have *3*: $\vdash$ (∃ *n*. *fpower* (*f* $\wedge$ *more*) *n*)^r = (∃ *n*. (*fpower* (*f* $\wedge$ *more*) *n*)^r)
by (*simp add*: *rev-exists*)
have *4*: $\vdash$ (∃ *n*. (*fpower* (*f* $\wedge$ *more*) *n*)^r) = (∃ *n*. (*fpower* ((*f* $\wedge$ *more*)^r) *n*))
by (*simp add*: *RevFPowerChop ExEqvRule*)
have *5*: $\vdash$ (*f* $\wedge$ *fmore*)^r = (*f*^{r} $\wedge$ *fmore*)
by (*metis* (*no-types*, *opaque-lifting*) *FiniteImpAnd RAnd RMoreEqvMore RevAndFinite*
inteq-reflection itl-def(*26*))
hence *6*: $\vdash$ (∃ *n*. (*fpower* ((*f* $\wedge$ *more*)^r) *n*)) = (∃ *n*. (*fpower* ((*f*^{r} $\wedge$ *more*)) *n*))
by (*metis AndMoreAndFiniteEqvAndFmore FPowerstar-WPowerstar FPowerstardef RAnd RMoreEqvMoreAlt*
fmore-d-def inteq-reflection rev-fun2-alt)
have *7*: $\vdash$ (∃ *n*. (*fpower* ((*f*^{r} $\wedge$ *more*)) *n*)) = *schopstar* (*f*^{r})
by (*simp add*: *fpowerstar-d-def schopstar-d-def*)
from *2 3 4 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *WRevAndFMore*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*f* $\wedge$ *fmore*) = (*wrev f* $\wedge$ *fmore*)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ *wrev* (*f* $\wedge$ *fmore*) = (*f* $\wedge$ *fmore*)^r
using *WRevAndFiniteAlt*[*of LIFT* (*f* $\wedge$ *fmore*)] **by** *auto*
have *2*: $\vdash$ *finite* $\longrightarrow$ (*f* $\wedge$ *fmore*)^r = (*f*^{r} $\wedge$ *fmore*)
by (*metis EqvReverseReverseAlt RMoreEqvMoreAlt fmore-d-def inteq-reflection rev-fun2*)
have *3*: $\vdash$ *finite* $\longrightarrow$ (*f*^{r} $\wedge$ *fmore*) = (*wrev f* $\wedge$ *fmore*)

by (*metis* (*no-types*, *lifting*) *2 AndMoreAndFiniteEqvAndFmore EqvReverseReverseAlt RAnd RMoreEqvMoreAlt WRevAndFiniteAlt fmore-d-def inteq-reflection*)
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 AndMoreAndFiniteEqvAndFmore WRAnd WRMoreEqvMoreAlt WRevAndFinite WRevAndFiniteAlt inteq-reflection*)
qed

lemma *WRevChopstar*:
$\vdash$ *wrev* (*schopstar f*) = (*finite* $\longrightarrow$ *schopstar* (*wrev f*))
proof −
have *1*: $\vdash$ (*schopstar f*) = ($\exists$ *n*. *fpower* (*f* $\wedge$ *more*) *n*)
by (*simp add*: *FPowerstardef schopstar-d-def*)
have *2*: $\vdash$ *wrev* (*schopstar f*) = *wrev* ($\exists$ *n*. *fpower* (*f* $\wedge$ *more*) *n*)
using *WREqvRule 1* **by** *blast*
have *3*: $\vdash$ *wrev* ($\exists$ *n*. *fpower* (*f* $\wedge$ *more*) *n*) = ($\exists$ *n*. *wrev* (*fpower* (*f* $\wedge$ *more*) *n*))
by (*simp add*: *wrev-exists*)
have *4*: $\vdash$ ($\exists$ *n*. *wrev* (*fpower* (*f* $\wedge$ *more*) *n*)) = ($\exists$ *n*. (*finite* $\longrightarrow$ *fpower* (*wrev* (*f* $\wedge$ *more*)) *n*))
by (*simp add*: *WRevFPowerChop ExEqvRule*)
have *5*: $\vdash$ *finite* $\longrightarrow$ *wrev* (*f* $\wedge$ *fmore*) = (*wrev f* $\wedge$ *fmore*)
by (*simp add*: *WRevAndFMore*)
hence *6*: $\vdash$ ($\exists$ *n*. (*finite* $\longrightarrow$ *fpower* (*wrev* (*f* $\wedge$ *more*)) *n*)) =
($\exists$ *n*. (*finite* $\longrightarrow$ *fpower* ((*wrev f* $\wedge$ *more*)) *n*))
by (*metis* (*no-types*, *lifting*) *ExEqvRule WRAnd WRMoreEqvMoreAlt WRevFPowerChop inteq-reflection*)
have *65*: $\vdash$ ($\exists$ *n*. (*finite* $\longrightarrow$ *fpower* ((*wrev f* $\wedge$ *more*)) *n*)) =
(*finite* $\longrightarrow$ ($\exists$ *n*. (*fpower* ((*wrev f* $\wedge$ *more*)) *n*)))
by *auto*
have *7*: $\vdash$ (*finite* $\longrightarrow$ ($\exists$ *n*. (*fpower* ((*wrev f* $\wedge$ *more*)) *n*))) =
(*finite* $\longrightarrow$ *schopstar* (*wrev f*))
by (*simp add*: *fpowerstar-d-def schopstar-d-def*)
from *2 3 4 6 7* **show** *?thesis*
by (*metis 65 inteq-reflection*)
qed

lemma *RevLen*:
$\vdash$ (*len k*)^r = *len k*
by (*auto simp add*: *Valid-def itl-defs freverse-d-def len-defs nlength-eq-enat-nfiniteD*)

lemma *WRevLen*:
$\vdash$ *wrev* (*len k*) = (*finite* $\longrightarrow$ *len k*)
by (*auto simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def len-defs nlength-eq-enat-nfiniteD*)

lemma *NotRevOmega*:
$\vdash$ $\neg$ *frev* (*omega f*)
by (*meson MP NotRevInfinite RImpRule lift-imp-neg omega-imp-inf*)

lemma *WRevOmega*:
$\vdash$ *wrev* (*omega f*) = *inf*
proof −
have *1*: $\vdash$ *wrev* (*omega f*) $\longrightarrow$ *inf*

by (*metis WRImpRule WRevInfinite inteq-reflection omega-imp-inf*)
have *2*: $\vdash$ *inf* $\longrightarrow$ *wrev* (*omega f*)
by (*simp add*: *freverse-d-def intI itl-defs*(*4*) *wfreverse-d-def*)
show *?thesis*
by (*simp add*: *1 2 Prop11*)
qed

lemmas *all-rev* = *rev-const rev-fun1 rev-fun2 rev-fun3 rev-fun1-alt rev-fun2-alt rev-fun3-alt rev-forall rev-exists rev-exists1 rev-current rev-next rev-penult rev-fin RevSkip RevChop RevChopAlt RevChopstar RevFinite RevAndFinite RevLen*

lemmas *all-wrev* = *wrev-fun1 wrev-fun2 wrev-fun3 wrev-fun1-alt wrev-fun2-alt wrev-fun3-alt wrev-forall wrev-exists wrev-exists1 WRevSkip WRevChop WRevChopAlt WRevChopstar WRevFinite WRevAndFinite WRevLen*

lemmas *all-rev-b* = *rev-fun1-alt rev-fun2-alt rev-fun3-alt rev-forall rev-exists RevChopAlt rev-exists1 RevSkip RevChopstar RevFinite RevAndFinite RevLen*

lemmas *all-wrev-b* = *wrev-fun1-alt wrev-fun2-alt wrev-fun3-alt wrev-forall wrev-exists WRevSkip WRevChopAlt WRevChopstar WRevAndFinite WRevLen*

lemmas *all-rev-unl* = *all-rev*[*THEN intD*]

lemmas *all-wrev-unl* = *all-wrev*[*THEN intD*]

lemmas *all-rev-eq* = *all-rev-b*[*THEN inteq-reflection*]

lemmas *all-wrev-eq* = *all-wrev-b*[*THEN inteq-reflection*]

3.12.4 Properties of Time Reversal

lemma *NotRev*:
$\vdash$ ($\neg$(f^r)) = (*wrev* ($\neg$*f*))
by (*simp add*: *wfreverse-d-def*)

lemma *NotWRev*:
$\vdash$ ($\neg$(*wrev f*)) = ($\neg$*f*)^r
by (*simp add*: *wfreverse-d-def*)

lemma *RNot*:
$\vdash$ *finite* $\longrightarrow$ ($\neg$*f*)^r = ($\neg$ f^r)
by (*simp add*: *rev-fun1*)

lemma *RNotAlt*:
$\vdash$ ($\neg$*f*)^r = (($\neg$ f^r) $\wedge$ *finite*)
by (*simp add*: *rev-fun1-alt*)

lemma *WRNot*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(\neg f) = (\neg\ (wrev\ f))$
by (*simp add*: *wrev-fun1*)

lemma *WRNotAlt*:
$\vdash$ *wrev* $(\neg f) = (finite \longrightarrow \neg\ (wrev\ f)\ )$
by (*simp add*: *wrev-fun1-alt*)

lemma *RRNot*:
$\vdash$ *finite* $\longrightarrow (\neg(f^r))^r = (\neg f)$
by (*simp add*: *freverse-d-def intI itl-defs*(5) *nfinite-nrev nrev-nrev*)

lemma *WRWRNot*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(\neg(wrev\ f)) = (\neg f)$
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverse EqvWReverseWReverseAlt WRevAndFiniteAlt WRevRev all-wrev-eq*(1) *all-wrev-eq*(9) *inteq-reflection*)

lemma *RRNotAlt*:
$\vdash (\neg(f^r))^r = (\neg f \wedge finite)$
by (*auto simp add*: *Valid-def freverse-d-def intI itl-defs*(5) *nfinite-nrev nrev-nrev*)

lemma *WRWRNotAlt*:
$\vdash$ *wrev* $(\neg(wrev\ f)) = (finite \longrightarrow \neg f)$
by (*metis EqvReverseReverseAlt EqvWReverseWReverseAlt WRevRev all-wrev-eq*(9) *int-simps*(4) *inteq-reflection wfreverse-d-def*)

lemma *RTrue*:
$\vdash$ *finite* $\longrightarrow (\#True)^r = \#True$
by (*metis int-simps*(1) *inteq-reflection rev-fun2*)

lemma *WRTrue*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(\#True) = \#True$
by (*metis int-simps*(1) *inteq-reflection wrev-fun2*)

lemma *RTrueAlt*:
$\vdash\ (\#True)^r = finite$
by (*simp add*: *freverse-d-def intI itl-defs*(5))

lemma *WRTrueAlt*:
$\vdash$ *wrev* $(\#True)$
by (*meson Prop11 Prop12 WReverseEqvAlt int-simps*(17))

lemma *ROr*:
$\vdash$ *finite* $\longrightarrow (f \vee g)^r = (f^r \vee g^r)$
by (*simp add*: *rev-fun2*)

lemma *WROr*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(f \vee g) = (wrev\ f \vee wrev\ g)$
by (*simp add*: *wrev-fun2*)

lemma *ROrAlt*:
$\vdash (f \vee g)^r = (f^r \vee g^r)$
by (*auto simp add*: *Valid-def freverse-d-def*)

lemma *WROrAlt*:
$\vdash$ *wrev* $(f \vee g) =$ (*wrev* $f \vee$ *wrev* g)
by (*auto simp add*: *Valid-def wfreverse-d-def freverse-d-def*)

lemma *RROr*:
$\vdash$ *finite* $\longrightarrow (f^r \vee g^r)^r = (f \vee g)$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow (f^r \vee g^r)^r = ((f^r)^r \vee (g^r)^r)$ **using** *ROr* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow ((f^r)^r \vee (g^r)^r) = (f \vee g)$ **using** *EqvReverseReverse*
by (*simp add*: *freverse-d-def intI itl-defs*(*5*) *nfinite-nrev nrev-nrev*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWROr*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*wrev* $f \vee$ *wrev* $g) = (f \vee g)$
by (*metis EqvWReverseWReverse WROrAlt inteq-reflection*)

lemma *RROrAlt*:
$\vdash \ (f^r \vee g^r)^r = ((f \vee g) \wedge$ *finite*)
by (*auto simp add*: *Valid-def freverse-d-def itl-defs nfinite-nrev nrev-nrev*)

lemma *WRWROrAlt*:
$\vdash$ *wrev* (*wrev* $f \vee$ *wrev* $g) =$ (*finite* $\longrightarrow f \vee g$)
by (*metis EqvWReverseWReverseAlt WROrAlt inteq-reflection*)

lemma *RRAnd*:
$\vdash$ *finite* $\longrightarrow (f^r \wedge g^r)^r = (f \wedge g)$
proof −
have *1*: $\vdash (f^r \wedge g^r)^r = ((f^r)^r \wedge (g^r)^r)$ **using** *RAnd* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow ((f^r)^r \wedge (g^r)^r) = (f \wedge g)$
by (*metis EqvReverseReverse RAnd inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRAnd*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*wrev* $f \wedge$ *wrev* $g) = (f \wedge g)$
by (*metis EqvWReverseWReverse WRAnd inteq-reflection*)

lemma *RRAndAlt*:
$\vdash \ (f^r \wedge g^r)^r = (f \wedge g \wedge$ *finite*)
by (*auto simp add*: *Valid-def freverse-d-def itl-defs nfinite-nrev nrev-nrev*)

lemma *WRWRAndAlt*:

$\vdash$ *wrev* (*wrev f* $\wedge$ *wrev g*) = (*finite* $\longrightarrow$ (*f* $\wedge$ *g*))
by (*metis EqvWReverseWReverseAlt WRAnd inteq-reflection*)

lemma *RAndEmptyEqvAndEmpty*:
$\vdash$ (*f* $\wedge$ *empty*)^r = (*f* $\wedge$ *empty*)
by (*simp add*: *Valid-def itl-defs freverse-d-def*)
(*metis ndropn-0 ndropn-nlast nlength-eq-enat-nfiniteD nrev-NNil the-enat-0 zero-enat-def*)

lemma *WRAndEmptyEqvAndEmpty*:
$\vdash$ *wrev* (*f* $\wedge$ *empty*) = (*finite* $\longrightarrow$ (*f* $\wedge$ *empty*))
by (*simp add*: *Valid-def itl-defs wfreverse-d-def freverse-d-def*)
(*metis ndropn-0 ndropn-nlast nrev-NNil the-enat-0*)

lemma *RNextEqvPrev*:
$\vdash$ *finite* $\longrightarrow$ ($\bigcirc$ *f*)^r = *prev* (*f*^{r})
by (*metis RevChop RevSkip inteq-reflection next-d-def prev-d-def*)

lemma *WRNextEqvPrev*:
$\vdash$ *finite* $\longrightarrow$ *wrev* ($\bigcirc$ *f*) = *prev* (*wrev f*)
proof −
have *3*: $\vdash$ *finite* $\longrightarrow$ *wrev f* ; (*finite* $\longrightarrow$ *skip*) = *wrev f* ; (*skip*)
by (*metis EqvWReverseWReverse EqvWReverseWReverseAlt FiniteRightChopEqvChop inteq-reflection*)

show *?thesis*
by (*metis 3 WRevRev all-rev-eq*(*8*) *all-wrev-eq*(*7*) *inteq-reflection next-d-def prev-d-def*)
qed

lemma *RNextEqvPrevAlt*:
$\vdash$ ($\bigcirc$ *f*)^r = *prev* (*f*^{r})
by (*metis RevChopAlt all-rev-eq*(*8*) *next-d-def prev-d-def*)

lemma *WRNextEqvPrevAlt*:
$\vdash$ *wrev* ($\bigcirc$ *f*) = (*finite* $\longrightarrow$ *prev* (*wrev f*))
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt WRevAndFiniteAlt WRevRev all-rev-eq*(*8*)
all-wrev-eq(*7*) *all-wrev-eq*(*9*) *inteq-reflection next-d-def prev-d-def*)

lemma *RRNextEqvPrev*:
$\vdash$ *finite* $\longrightarrow$($\bigcirc$ (*f*^{r}))^r = *prev* (*f*)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$($\bigcirc$ (*f*^{r}))^r = *prev* ((*f*^{r})^r) **using** *RNextEqvPrev* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$ (*f*^{r})^r = *f*
by (*simp add*: *EqvReverseReverse*)
have *3*: $\vdash$ *finite* $\longrightarrow$ *prev* ((*f*^{r})^r) = *prev f*
unfolding *prev-d-def* **using** *2* **by** (*simp add*: *FiniteChopEqv*)
from *1 3* **show** *?thesis* **using** *FiniteTransEqv* **by** *blast*
qed

lemma *WRWRNextEqvPrev*:
$\vdash$ *finite* $\longrightarrow$ *wrev* ($\bigcirc$ (*wrev f*)) = *prev* (*f*)

by (*metis EqvWReverseWReverse FiniteChopEqv FiniteTransEqv WRNextEqvPrev prev-d-def*)

lemma *RRNextEqvPrevAlt*:
$\vdash (\bigcirc (f^r))^r = (\mathit{finite} \wedge \mathit{prev}\ f\ )$
by (*metis EqvReverseReverseAlt RevChopAlt all-rev-eq*(*8*) *inteq-reflection lift-and-com next-d-def prev-d-def*)

lemma *WRWRNextEqvPrevAlt*:
$\vdash$ *wrev* $(\bigcirc$ (*wrev f*)) = (*finite* $\longrightarrow$ *prev f*)
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt WRevAndFiniteAlt all-rev-eq*(*8*)
all-wrev-eq(*7*) *all-wrev-eq*(*9*) *inteq-reflection next-d-def prev-d-def*)

lemma *RWNextEqvWPrev*:
$\vdash$ *finite* $\longrightarrow$ $(\mathit{wnext}\ f)^r = \mathit{wprev}(f^r)$
proof −
have *1*: $\vdash$ *finite*$\longrightarrow$$(\mathit{wnext}\ f)^r = (\mathit{empty} \vee \mathit{next}\ f)^r$
by (*metis ROr ROrAlt WnextEqvEmptyOrNext inteq-reflection*)
have *2*: $\vdash$ *finite* $\longrightarrow$ $(\mathit{empty} \vee \mathit{next}\ f)^r = (\mathit{empty}^r \vee (\mathit{next}\ f)^r)$
by (*simp add*: *ROr*)
have *3*: $\vdash$ *finite* $\longrightarrow$ $(\mathit{empty}^r \vee (\mathit{next}\ f)^r) = (\mathit{empty} \vee (\mathit{prev}\ (f^r)))$
using *REmptyEqvEmpty RNextEqvPrev*[*of f*] **by** *fastforce*
have *4*: $\vdash$ *finite* $\longrightarrow$ $(\mathit{empty} \vee (\mathit{prev}\ (f^r))) = \mathit{wprev}(f^r)$
by (*metis WprevEqvEmptyOrPrev int-simps*(*1*) *int-simps*(*12*) *inteq-reflection*)
show *?thesis*
by (*metis 3 FiniteTransEqv WnextEqvEmptyOrNext WprevEqvEmptyOrPrev inteq-reflection rev-fun2*)
qed

lemma *WRWNextEqvWPrev*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(\mathit{wnext}\ f) = \mathit{wprev}(\mathit{wrev}\ f)$
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverse EqvWReverseWReverseAlt FiniteTransEqv*
RNextEqvPrevAlt WRevRev int-simps(*4*) *inteq-reflection wfreverse-d-def wnext-d-def*
wprev-d-def wrev-fun1)

lemma *RWNextEqvWPrevAlt*:
$\vdash (\mathit{wnext}\ f)^r = \mathit{wprev}(f^r)$
proof −
have *1*: $\vdash (\mathit{wnext}\ f)^r = (\mathit{empty} \vee \mathit{next}\ f)^r$
by (*simp add*: *REqvRule WnextEqvEmptyOrNext*)
have *2*: $\vdash$ $(\mathit{empty} \vee \mathit{next}\ f)^r = (\mathit{empty}^r \vee (\mathit{next}\ f)^r)$
by (*simp add*: *ROrAlt*)
have *3*: $\vdash$ $(\mathit{empty}^r \vee (\mathit{next}\ f)^r) = (\mathit{empty} \vee (\mathit{prev}\ (f^r)))$
by (*metis 2 REmptyEqvEmpty RNextEqvPrevAlt inteq-reflection*)
have *4*: $\vdash$ $(\mathit{empty} \vee (\mathit{prev}\ (f^r))) = \mathit{wprev}(f^r)$
by (*meson WprevEqvEmptyOrPrev int-iffD1 int-iffD2 int-iffI*)
show *?thesis*
by (*metis 1 2 3 4 inteq-reflection*)
qed

lemma *WRWNextEqvWPrevAlt*:
$\vdash$ *wrev* $(\mathit{wnext}\ f) = \mathit{wprev}(\mathit{wrev}\ f)$

by (*metis* (*no-types*, *lifting*) *RNextEqvPrevAlt int-simps*(*4*) *inteq-reflection itl-def*(*20*) *wfreverse-d-def wprev-d-def*)

lemma *RRWNextEqvWPrev*:
$\vdash$ *finite* $\longrightarrow$ $(wnext\ (f^r))^r = wprev(f)$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow (wnext\ (f^r))^r = wprev\ ((f^r)^r)$ **using** *RWNextEqvWPrev* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow (f^r)^r = f$
by (*simp add*: *EqvReverseReverse*)
have *3*: $\vdash$ *finite* $\longrightarrow wprev\ ((f^r)^r) = wprev\ f$
by (*simp add*: *2 FiniteWPrevEqv*)
from *1 3* **show** *?thesis* **using** *FiniteTransEqv* **by** *blast*
qed

lemma *WRWRWNextEqvWPrev*:
$\vdash$ *finite* $\longrightarrow$ $wrev\ (wnext\ (wrev\ f)) = wprev(f)$
using *EqvWReverseWReverse FiniteTransEqv FiniteWPrevEqv WRWNextEqvWPrev* **by** *blast*

lemma *RRWNextEqvWPrevAlt*:
$\vdash$ $(wnext\ (f^r))^r = (finite \wedge wprev\ f)$
by (*metis FiniteImpAnd ROrAlt RRWNextEqvWPrev WnextEqvEmptyOrNext all-rev-eq*(*2*) *inteq-reflection lift-and-com*)

lemma *WRWRWNextEqvWPrevAlt*:
$\vdash$ $wrev\ (wnext\ (wrev\ f)) = (finite \longrightarrow wprev(f))$
using *WRWRWNextEqvWPrev WRevRev* **by** *fastforce*

lemma *RPrevEqvNext*:
$\vdash$ *finite* $\longrightarrow$ $(prev\ f)^r = \bigcirc\ (f^r)$
by (*metis RevChop RevSkip inteq-reflection next-d-def prev-d-def*)

lemma *WRPrevEqvNext*:
$\vdash$ *finite* $\longrightarrow$ $wrev\ (prev\ f) = \bigcirc\ (wrev\ f)$
by (*metis* (*no-types*, *opaque-lifting*) *EqvWReverseWReverse EqvWReverseWReverseAlt FiniteChopEqv WRevRev all-rev-eq*(*8*) *all-wrev-eq*(*7*) *inteq-reflection next-d-def prev-d-def*)

lemma *RPrevEqvNextAlt*:
$\vdash$ $(prev\ f)^r = \bigcirc\ (f^r)$
by (*metis RevChopAlt all-rev-eq*(*8*) *next-d-def prev-d-def*)

lemma *WRPrevEqvNextAlt*:
$\vdash$ $wrev\ (prev\ f) = (finite \longrightarrow \bigcirc\ (wrev\ f))$
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt WRWRNextEqvPrevAlt WRevAndFiniteAlt WRevRev*

all-rev-eq(*10*) *all-rev-eq*(*2*) *inteq-reflection*)

lemma *RRPrevEqvNext*:
$\vdash$ *finite* $\longrightarrow$ $(prev\ (f^r))^r = \bigcirc\ (f)$
proof −

have *1*: $\vdash$ *finite*$\longrightarrow$$(prev\ (f^r))^r = \bigcirc\ ((f^r)^r)$ **using** *RPrevEqvNext* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$$\bigcirc\ ((f^r)^r) = \bigcirc\ f$
by (*simp add*: *EqvReverseReverse FiniteRightChopEqvChop next-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRPrevEqvNext*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*prev* (*wrev f*)) = $\bigcirc$ (*f*)
by (*metis EqvWReverseWReverse FiniteRightChopEqvChop FiniteTransEqv WRPrevEqvNext next-d-def*)

lemma *RRPrevEqvNextAlt*:
$\vdash$ $(prev\ (f^r))^r = (finite \wedge \bigcirc\ f)$
by (*metis EqvReverseReverseAlt RNextEqvPrevAlt inteq-reflection lift-and-com*)

lemma *WRWRPrevEqvNextAlt*:
$\vdash$ *wrev* (*prev* (*wrev f*)) = (*finite* $\longrightarrow$ $\bigcirc$ *f*)
by (*metis EqvWReverseWReverseAlt WRNextEqvPrevAlt WRPrevEqvNextAlt WRWRNextEqvPrevAlt inteq-reflection*)

lemma *RWPrevEqvWNext*:
$\vdash$ *finite* $\longrightarrow$$(wprev\ f)^r = wnext(f^r)$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ $(wprev\ f)^r = (empty \vee prev\ f)^r$
by (*metis WprevEqvEmptyOrPrev int-simps*(*1*) *int-simps*(*12*) *inteq-reflection*)
have *2*: $\vdash$ *finite* $\longrightarrow$ $(empty \vee prev\ f)^r = (empty^r \vee (prev\ f)^r)$
by (*simp add*: *ROr*)
have *3*: $\vdash$ *finite* $\longrightarrow$ $(prev\ f)^r = (next\ (f^r))$
by (*simp add*: *RPrevEqvNext*)
have *4*: $\vdash$ *finite* $\longrightarrow$ $empty^r = empty$
using *REmptyEqvEmpty* **by** *auto*
have *5*: $\vdash$ *finite* $\longrightarrow$ $(empty^r \vee (prev\ f)^r) = (empty \vee (next\ (f^r)))$
by (*simp add*: *3 4 FiniteOrEqv*)
have *6*: $\vdash$ *finite* $\longrightarrow$ $(empty \vee (next\ (f^r))) = wnext(f^r)$
by (*metis WnextEqvEmptyOrNext int-simps*(*1*) *int-simps*(*12*) *inteq-reflection*)
show *?thesis*
by (*metis 5 FiniteTransEqv WnextEqvEmptyOrNext WprevEqvEmptyOrPrev inteq-reflection rev-fun2*)
qed

lemma *WRWPrevEqvWNext*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*wprev f*) = *wnext*(*wrev f*)
by (*metis* (*no-types*, *opaque-lifting*) *RPrevEqvNextAlt WRTrueAlt WReverseEqvAlt int-simps*(*1*)
int-simps(*4*) *inteq-reflection wfreverse-d-def wnext-d-def wprev-d-def*)

lemma *RWPrevEqvWNextAlt*:
$\vdash$ $(wprev\ f)^r = wnext(f^r)$
proof −
have *1*: $\vdash$ $(wprev\ f)^r = (empty \vee prev\ f)^r$
by (*metis WprevEqvEmptyOrPrev int-simps*(*1*) *inteq-reflection*)
have *2*: $\vdash$ $(empty \vee prev\ f)^r = (empty^r \vee (prev\ f)^r)$

by (*simp add*: *ROrAlt*)
have *3*: ⊢ $(prev\ f)^r = (next\ (f^r))$
by (*simp add*: *RPrevEqvNextAlt*)
have *4*: ⊢ $empty^r = empty$
using *REmptyEqvEmpty* **by** *auto*
have *5*: ⊢ $(empty^r \vee (prev\ f)^r) = (empty \vee (next\ (f^r)))$
by (*metis 2 3 REmptyEqvEmpty inteq-reflection*)
have *6*: ⊢ $(empty \vee (next\ (f^r))) = wnext(f^r)$
by (*metis WnextEqvEmptyOrNext inteq-reflection*)
show *?thesis*
by (*metis 1 2 5 6 inteq-reflection*)
qed

lemma *WRWPrevEqvWNextAlt*:
⊢ $wrev\ (wprev\ f) = (finite \longrightarrow wnext(wrev\ f))$
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt RPrevEqvNextAlt WRWRNotAlt WRevAndFiniteAlt int-simps*(*4*) *inteq-reflection wfreverse-d-def wnext-d-def wprev-d-def*)

lemma *RRWPrevEqvWNext*:
⊢ $finite \longrightarrow (wprev\ (f^r))^r = wnext(f)$
proof −
have *1*: ⊢ $finite \longrightarrow (wprev\ (f^r))^r = wnext\ ((f^r)^r)$ **using** *RWPrevEqvWNext* **by** *blast*
have *2*: ⊢ $finite \longrightarrow wnext\ ((f^r)^r) = wnext\ f$
by (*simp add*: *EqvReverseReverse FiniteWNextEqv*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRWPrevEqvWNext*:
⊢ $finite \longrightarrow wrev\ (wprev\ (wrev\ f)) = wnext(f)$
by (*metis EqvWReverseWReverse WRWNextEqvWPrevAlt inteq-reflection*)

lemma *RRWPrevEqvWNextAlt*:
⊢ $(wprev\ (f^r))^r = (finite \wedge wnext\ f)$
by (*meson EqvReverseReverseAlt Prop04 REqvRule RWNextEqvWPrevAlt lift-and-com*)

lemma *WRWRWPrevEqvWNextAlt*:
⊢ $wrev\ (wprev\ (wrev\ f)) = (finite \longrightarrow wnext\ f)$
by (*metis EqvWReverseWReverseAlt WRWNextEqvWPrevAlt inteq-reflection*)

lemma *RDiamondEqvDi*:
⊢ $finite \longrightarrow (\Diamond f)^r = di\ (f^r)$
by (*metis FiniteRightChopEqvChop FiniteTransEqv Prop11 RevChop all-rev-eq*(*10*) *di-d-def int-simps*(*9*) *sometimes-d-def*)

lemma *WRDiamondEqvDi*:
⊢ $finite \longrightarrow wrev\ (\Diamond f) = di\ (wrev\ f)$
unfolding *di-d-def sometimes-d-def*
by (*metis FiniteRightChopEqvChop FiniteTransEqv Prop11 WRevChop WReverseEqv int-simps*(*9*)

lift-imp-trans)

lemma *RDiamondEqvDiAlt*:
$\vdash \ (\Diamond f)^r = \ (f^r)$;*finite*
by (*metis RevChopAlt all-rev-eq*(*10*) *sometimes-d-def*)

lemma *WRDiamondEqvDiAlt*:
$\vdash$ *wrev* $(\Diamond f)$ = *di* (*wrev f*)
unfolding *di-d-def sometimes-d-def*
by (*metis WRevRev all-rev-eq*(*10*) *all-wrev-eq*(*7*) *int-simps*(*13*) *inteq-reflection*)

lemma *RRDiamondEqvDi*:
$\vdash$ *finite* $\longrightarrow (\Diamond(f^r))^r$ = *di* (*f*)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow (\Diamond\ (f^r))^r$ = *di* $((f^r)^r)$ **using** *RDiamondEqvDi* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$*di* $((f^r)^r)$ = *di f*
by (*simp add*: *EqvReverseReverse FiniteChopEqv di-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRDiamondEqvDi*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(\Diamond$(*wrev f*)) = *di* (*f*)
by (*metis EqvWReverseWReverse FiniteChopEqv FiniteTransEqv WRDiamondEqvDi di-d-def*)

lemma *RRDiamondEqvDiAlt*:
$\vdash \ (\Diamond(f^r))^r = \ ($*finite* $\wedge$ *di f*)
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt FiniteImpAnd RRAndAlt RRDiamondEqvDi all-rev-eq*(*10*) *all-rev-eq*(*6*) *inteq-reflection lift-and-com sometimes-d-def*)

lemma *WRWRDiamondEqvDiAlt*:
$\vdash$ *wrev* $(\Diamond$(*wrev f*)) = (*finite* $\longrightarrow$ *di f*)
using *WRWRDiamondEqvDi WRevRev* **by** *fastforce*

lemma *RBoxEqvBi*:
$\vdash$ *finite* $\longrightarrow (\Box\ f)^r$ = *bi* (f^r)
unfolding *always-d-def bi-d-def*
by (*metis FiniteChopEqv FiniteNotEqv FiniteTransEqv RDiamondEqvDi RNot di-d-def*)

lemma *WRBoxEqvBi*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(\Box\ f)$ = *bi* (*wrev f*)
unfolding *always-d-def bi-d-def*
by (*simp add*: *FiniteNotEqv RDiamondEqvDi wfreverse-d-def*)

lemma *RBoxEqvBiAlt*:
$\vdash \ (\Box\ f)^r = (bi\ (f^r) \wedge$ *finite*)
by (*metis FiniteImpAnd RAnd RBoxEqvBi RTrueAlt int-simps*(*17*) *inteq-reflection lift-and-com*)

lemma *WRBoxEqvBiAlt*:
$\vdash$ *wrev* $(\Box\ f)$ = (*finite* $\longrightarrow$ *bi* (*wrev f*))

using *WRBoxEqvBi WRevRev* **by** *fastforce*

lemma *RRBoxEqvBi*:
$\vdash$ *finite* $\longrightarrow$ $(\Box\ (f^r))^r$ = *bi* (f)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ $(\Box\ (f^r))^r$ = *bi* $((f^r)^r)$ **using** *RBoxEqvBi* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$ *bi* $((f^r)^r)$ = *bi f*
by (*simp add*: *EqvReverseReverse FiniteChopEqv FiniteNotEqv bi-d-def di-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRBoxEqvBi*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(\Box\ (wrev\ f))$ = *bi* (f)
by (*metis* (*no-types*, *opaque-lifting*) *EqvWReverseWReverse FiniteChopEqv FiniteNotEqv FiniteTransEqv WRBoxEqvBi bi-d-def di-d-def*)

lemma *RRBoxEqvBiAlt*:
$\vdash$ $(\Box\ (f^r))^r$ = (*bi f* $\wedge$ *finite*)
by (*metis FiniteImpAnd RAnd RRBoxEqvBi RTrueAlt int-simps*(*17*) *inteq-reflection lift-and-com*)

lemma *WRWRBoxEqvBiAlt*:
$\vdash$ *wrev* $(\Box\ (wrev\ f))$ = (*finite* $\longrightarrow$ *bi f*)
using *WRWRBoxEqvBi WRevRev* **by** *fastforce*

lemma *RDiEqvDiamond*:
$\vdash$ *finite* $\longrightarrow$ $(di\ f)^r$ = $\Diamond$ (f^r)
unfolding *sometimes-d-def di-d-def*
by (*meson FiniteChopEqv FiniteTransEqv Prop11 RTrue RevChop int-simps*(*8*))

lemma *WRDiEqvDiamond*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(di\ f)$ = $\Diamond$ $(wrev\ f)$
by (*metis* (*no-types*, *opaque-lifting*) *EqvWReverseWReverseAlt RTrueAlt WRevAndFiniteAlt all-rev-eq*(*2*) *all-rev-eq*(*6*) *all-wrev-eq*(*2*) *all-wrev-eq*(*9*) *di-d-def int-simps*(*1*) *int-simps*(*12*) *inteq-reflection sometimes-d-def*)

lemma *RDiEqvDiamondAlt*:
$\vdash$ $(di\ f)^r$ = $\Diamond$ (f^r)
by (*metis RTrueAlt RevChopAlt di-d-def inteq-reflection sometimes-d-def*)

lemma *WRDiEqvDiamondAlt*:
$\vdash$ *wrev* $(di\ f)$ = (*finite* $\longrightarrow$ $\Diamond$ $(wrev\ f)$)
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt WRWRDiamondEqvDiAlt WRevAndFiniteAlt WRevRev all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RRDiEqvDiamond*:
$\vdash$ *finite* $\longrightarrow$ $(di\ (f^r))^r$ = $\Diamond$ (f)

proof −
have *1*: ⊢ *finite* ⟶ (*di* (*f*^{r}))^r = ◇ ((*f*^{r})^r) **using** *RDiEqvDiamond* **by** *blast*
have *2*: ⊢ *finite* ⟶ ◇ ((*f*^{r})^r) = ◇ *f*
by (*simp add*: *EqvReverseReverse FiniteRightChopEqvChop sometimes-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRDiEqvDiamond*:
⊢ *finite* ⟶ *wrev* (*di* (*wrev f*)) = ◇ (*f*)
by (*metis EqvWReverseWReverse WRDiamondEqvDiAlt int-eq*)

lemma *RRDiEqvDiamondAlt*:
⊢ (*di* (*f*^{r}))^r = (*finite* ∧ ◇ *f*)
by (*metis EqvReverseReverseAlt RDiEqvDiamondAlt all-rev-eq*(*10*) *all-rev-eq*(*6*) *inteq-reflection lift-and-com sometimes-d-def*)

lemma *WRWRDiEqvDiamondAlt*:
⊢ *wrev* (*di* (*wrev f*)) = (*finite* ⟶ ◇ *f*)
by (*meson EqvWReverseWReverseAlt Prop04 WRDiamondEqvDiAlt WREqvRule*)

lemma *RBiEqvBox*:
⊢ *finite* ⟶(*bi f*)^r = □ (*f*^{r})
unfolding *always-d-def bi-d-def*
by (*metis FiniteNotEqv FiniteRightChopEqvChop FiniteTransEqv RDiEqvDiamond RNot sometimes-d-def*)

lemma *WRBiEqvBox*:
⊢ *finite* ⟶ *wrev* (*bi f*) = □ (*wrev f*)
unfolding *always-d-def bi-d-def*
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverse EqvReverseReverseAlt FiniteTransEqv RDiEqvDiamondAlt all-rev-eq*(*1*) *int-simps*(*4*) *inteq-reflection wfreverse-d-def wrev-fun1*)

lemma *RBiEqvBoxAlt*:
⊢ (*bi f*)^r = (□ (*f*^{r}) ∧ *finite*)
by (*metis FiniteImpAnd RAnd RBiEqvBox RTrueAlt int-simps*(*17*) *inteq-reflection lift-and-com*)

lemma *WRBiEqvBoxAlt*:
⊢ *wrev* (*bi f*) = (*finite* ⟶ □ (*wrev f*))
using *WRBiEqvBox WRevRev* **by** *fastforce*

lemma *RRBiEqvBox*:
⊢ *finite* ⟶ (*bi* (*f*^{r}))^r = □ (*f*)
proof −
have *1*: ⊢ *finite* ⟶ (*bi* (*f*^{r}))^r = □ ((*f*^{r})^r) **using** *RBiEqvBox* **by** *blast*
have *2*: ⊢ *finite*⟶□ ((*f*^{r})^r) = □ *f*
by (*simp add*: *EqvReverseReverse FiniteNotEqv FiniteRightChopEqvChop always-d-def sometimes-d-def*)

from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRBiEqvBox*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*bi* (*wrev f*)) = $\Box$ (*f*)
by (*metis* (*no-types*, *opaque-lifting*) *EqvWReverseWReverse FiniteNotEqv FiniteRightChopEqvChop FiniteTransEqv WRBiEqvBox always-d-def sometimes-d-def*)

lemma *RRBiEqvBoxAlt*:
$\vdash$ (*bi* (f^r))^r = ($\Box$ *f* $\wedge$ *finite*)
by (*meson FiniteImpAnd Prop04 Prop11 RBiEqvBox RBiEqvBoxAlt RRBiEqvBox*)

lemma *WRWRBiEqvBoxAlt*:
$\vdash$ *wrev* (*bi* (*wrev f*)) = (*finite* $\longrightarrow$ $\Box$ *f*)
by (*metis EqvWReverseWReverseAlt WRBiEqvBoxAlt WRBoxEqvBiAlt WRWRBoxEqvBiAlt inteq-reflection*)

lemma *RDaEqvDa*:
$\vdash$ *finite* $\longrightarrow$ (*da f*)^r = *da*(f^r)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ (*finite*;(*f*;#*True*))^r = (*f*;#*True*)^r; *finite*^{r} **using** *RevChop* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$ (*f*;#*True*)^r; *finite*^{r} = (*f*;#*True*)^r; *finite*
by (*simp add*: *all-rev-eq*(*10*))
have *3*: $\vdash$ *finite* $\longrightarrow$ (*f*;#*True*)^r; *finite* = (#*True*^{r};f^r);*finite*
by (*simp add*: *FiniteChopEqv RevChop*)
have *4*: $\vdash$ *finite* $\longrightarrow$(#*True*^{r};f^r);*finite* = (#*True*;f^r);*finite*
using *FiniteChopEqv RTrue* **by** *blast*
have *5*: $\vdash$ *finite* $\longrightarrow$(#*True*;f^r);*finite* = #*True*;(f^r;*finite*)
using *ChopAssoc* **by** *fastforce*
have *6*: $\vdash$ *finite* $\longrightarrow$(*finite*;(*f*;#*True*))^r = #*True*;(f^r;*finite*) **using** *1 2 3 4 5* **by** *fastforce*
have *7*: $\vdash$ *finite* $\longrightarrow$ #*True*;(f^r;*finite*) = *finite*;(f^r;#*True*)
by (*meson FiniteChopEqv FiniteRightChopEqvChop FiniteTransEqv Prop11 int-simps*(*8*) *int-simps*(*9*))
from *6 7* **show** *?thesis*
unfolding *da-d-def*
using *FiniteTransEqv* **by** *blast*
qed

lemma *WRDaEqvDa*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*da f*) = *da*(*wrev f*)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ *wrev* (*finite*;(*f*;#*True*)) = *wrev* (*f*;#*True*); *wrev finite*
by (*simp add*: *WRevChop*)
have *2*: $\vdash$ *finite* $\longrightarrow$ *wrev* (*f*;#*True*); *wrev finite* = *wrev* (*f*;#*True*); *finite*
by (*metis FiniteRightChopEqvChop InfEqvNotFinite all-rev-eq*(*10*) *finite-d-def inteq-reflection wfreverse-d-def wrev-fun1*)
have *3*: $\vdash$ *finite* $\longrightarrow$ *wrev* (*f*;#*True*) ; *finite* = (*wrev* #*True*;*wrev f*);*finite*
by (*simp add*: *FiniteChopEqv WRevChop*)
have *4*: $\vdash$ *finite* $\longrightarrow$ (*wrev* #*True*;*wrev f*);*finite* = (#*True*;*wrev f*);*finite*
using *FiniteChopEqv WRTrue* **by** *blast*
have *5*: $\vdash$ *finite* $\longrightarrow$ (#*True*;*wrev f*);*finite* = #*True*;(*wrev f*;*finite*)

using *ChopAssoc* **by** *fastforce*
have *6*: $\vdash$ *finite* $\longrightarrow$ *wrev* (*finite*;(*f*;#*True*)) = #*True*;(*wrev f*;*finite*)
using *1 2 3 4 5* **by** *fastforce*
have *7*: $\vdash$ *finite* $\longrightarrow$ #*True*;(*wrev f*;*finite*) = *finite*;(*wrev f*;#*True*)
by (*meson FiniteChopEqv FiniteRightChopEqvChop FiniteTransEqv Prop11 int-simps*(*8*) *int-simps*(*9*))
from *6 7* **show** *?thesis*
by (*metis FiniteTransEqv da-d-def*)
qed

lemma *RDaEqvDaAlt*:
$\vdash$ (*da f*)^r = (*finite* $\wedge$ *da* (*f*^{r}))
by (*metis* (*no-types*, *lifting*) *EqvReverseReverseAlt FiniteImpAnd RDaEqvDa RRAndAlt WRevAndFiniteAlt*

inteq-reflection lift-and-com)

lemma *WRDaEqvDaAlt*:
$\vdash$ *wrev* (*da f*) = (*finite* $\longrightarrow$ *da* (*wrev f*))
by (*metis FiniteImp FiniteImpAnd Prop10 WRDaEqvDa WRevAndFiniteAlt WRevRev int-iffD1 inteq-reflection*)

lemma *RRDaEqvDa*:
$\vdash$ *finite* $\longrightarrow$ (*da* (*f*^{r}))^r = *da*(*f*)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ (*da* (*f*^{r}))^r = *da* ((*f*^{r})^r) **using** *RDaEqvDa* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$ *da* ((*f*^{r})^r) = *da f*
by (*simp add*: *EqvReverseReverse FiniteChopEqv FiniteRightChopEqvChop da-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRDaEqvDa*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*da* (*wrev f*)) = *da*(*f*)
by (*metis EqvWReverseWReverse FiniteChopEqv FiniteRightChopEqvChop FiniteTransEqv WRDaEqvDa da-d-def*)

lemma *RRDaEqvDaAlt*:
$\vdash$ (*da* (*f*^{r}))^r = (*finite* $\wedge$ *da f*)
by (*metis FiniteImpAnd RDaEqvDa RDaEqvDaAlt RRDaEqvDa inteq-reflection lift-and-com*)

lemma *WRWRDaEqvDaAlt*:
$\vdash$ *wrev* (*da* (*wrev f*)) = (*finite* $\longrightarrow$ *da f*)
by (*metis EqvWReverseWReverseAlt WRDaEqvDaAlt WRevFiniteImp inteq-reflection*)

lemma *RBaEqvBa*:
$\vdash$ *finite* $\longrightarrow$ (*ba f*)^r = *ba*(*f*^{r})
unfolding *ba-d-def*
by (*metis FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop FiniteTransEqv RDaEqvDa RNot itl-def*(*12*))

lemma *WRBaEqvBa*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*ba f*) = *ba*(*wrev f*)
unfolding *ba-d-def wfreverse-d-def*

by (*simp add*: *FiniteNotEqv RDaEqvDa*)

lemma *RBaEqvBaAlt*:
$\vdash \ (ba\ f)^r = (ba(f^r) \wedge finite)$
by (*metis FiniteImpAnd RAnd RBaEqvBa RTrueAlt int-simps*(*17*) *inteq-reflection lift-and-com*)

lemma *WRBaEqvBaAlt*:
$\vdash \ wrev\ (ba\ f) = (finite \longrightarrow ba(wrev\ f))$
by (*metis* (*no-types*, *opaque-lifting*) *EqvWReverseWReverseAlt FiniteImpAnd RDaEqvDa WRevAndFiniteAlt*

WRevRev all-wrev-eq(*9*) *ba-d-def int-simps*(*4*) *inteq-reflection wfreverse-d-def*)

lemma *RRBaEqvBa*:
$\vdash finite \longrightarrow (ba\ (f^r))^r = ba(f)$
proof −
have *1*: $\vdash finite \longrightarrow (ba\ (f^r))^r = ba\ ((f^r)^r)$ **using** *RBaEqvBa* **by** *blast*
have *2*: $\vdash finite \longrightarrow ba\ ((f^r)^r) = ba\ f$
by (*simp add*: *EqvReverseReverse FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop ba-d-def da-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRBaEqvBa*:
$\vdash finite \longrightarrow wrev\ (ba\ (wrev\ f)) = ba(f)$
by (*metis* (*no-types*, *opaque-lifting*) *EqvWReverseWReverse FiniteChopEqv FiniteNotEqv*
FiniteRightChopEqvChop FiniteTransEqv WRBaEqvBa ba-d-def da-d-def)

lemma *RRBaEqvBaAlt*:
$\vdash \ (ba\ (f^r))^r = (ba\ f \wedge finite)$
by (*meson FiniteImpAnd Prop04 Prop11 RBaEqvBa RBaEqvBaAlt RRBaEqvBa*)

lemma *WRWRBaEqvBaAlt*:
$\vdash \ wrev\ (ba\ (wrev\ f)) = (finite \longrightarrow ba\ f\ )$
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt EqvWReverseWReverseAlt FiniteImpAnd*
WRWRBaEqvBa all-wrev-eq(*9*) *int-simps*(*4*) *inteq-reflection wfreverse-d-def*)

lemma *RInitEqvFin*:
$\vdash finite \longrightarrow (init\ f)^r = fin(f)$
proof −
have *1*: $\vdash finite \longrightarrow (init\ f)^r = ((f \wedge empty);\#True)^r$
by (*metis init-d-def int-simps*(*1*) *int-simps*(*12*) *inteq-reflection lift-and-com*)
have *2*: $\vdash finite \longrightarrow ((f \wedge empty);\#True)^r = (\#True;(f \wedge empty)^r)$
by (*meson FiniteChopEqv FiniteTransEqv RTrue RevChop*)
have *3*: $\vdash \#True;(f \wedge empty)^r = \#True;(f^r \wedge empty)$
by (*metis RAnd REmptyEqvEmpty RightChopEqvChop int-eq*)
have *4*: $\vdash finite \longrightarrow \#True;(f^r \wedge empty) = \#True;(f \wedge empty)$
using *RAndEmptyEqvAndEmpty*
by (*metis 3 FiniteRightChopEqvChop REmptyEqvEmpty inteq-reflection rev-fun2*)
have *5*: $\vdash \#True;(f \wedge empty) = fin(f)$
using *FinEqvTrueChopAndEmpty* **by** *fastforce*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*

qed

lemma *WRInitEqvFin*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*init f*) = *fin*(*f*)
by (*metis EqvWReverseWReverse EqvWReverseWReverseAlt FiniteTransEqv RInitEqvFin WRevRev inteq-reflection*)

lemma *RInitEqvFinAlt*:
$\vdash$ $(init\ f)^r = (sfin(f))$
proof −
have *1*: $\vdash$ $(init\ f)^r = ((f \wedge empty);\#True)^r$
by (*metis init-d-def int-simps*(*1*) *inteq-reflection lift-and-com*)
have *2*: $\vdash$ $((f \wedge empty);\#True)^r = (finite;(f \wedge empty)^r)$
by (*metis RTrueAlt RevChopAlt inteq-reflection*)
have *3*: $\vdash$ $finite;(f \wedge empty)^r = finite;(f^r \wedge empty)$
by (*metis RAnd REmptyEqvEmpty RightChopEqvChop int-eq*)
have *4*: $\vdash$ $finite;(f^r \wedge empty) = finite;(f \wedge empty)$
by (*metis 3 RAndEmptyEqvAndEmpty inteq-reflection*)
have *5*: $\vdash$ $finite;(f \wedge empty) = sfin(f)$
by (*metis DiamondSChopdef FiniteChopEqvDiamond SFinEqvTrueSChopAndEmpty inteq-reflection*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *WRInitEqvFinAlt*:
$\vdash$ *wrev* (*init f*) = (*finite* $\longrightarrow$ *fin*(*f*))
by (*metis EqvWReverseWReverseAlt FiniteImpAnd RInitEqvFin WRevRev all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RFinEqvInit*:
$\vdash$ *finite* $\longrightarrow$ $(fin\ f)^r = init\ (f)$
proof −
have *1*: $\vdash$ $fin\ f = \#True;(f \wedge empty)$
using *FinEqvTrueChopAndEmpty* **by** *auto*
have *2*: $\vdash$ $(fin\ f)^r = (\#True;(f \wedge empty))^r$
using *1 REqvRule* **by** *blast*
have *3*: $\vdash$ $finite \longrightarrow (\#True;(f \wedge empty))^r = (f \wedge empty)^r;\#True$
by (*meson FiniteRightChopEqvChop FiniteTransEqv RTrue RevChop*)
have *4*: $\vdash$ $(f \wedge empty)^r;\#True = (f^r \wedge empty);\#True$
using *LeftChopEqvChop RAnd REmptyEqvEmpty* **by** (*metis int-eq*)
have *5*: $\vdash$ $(f \wedge empty)^r;\#True = (f \wedge empty);\#True$
by (*simp add*: *RAndEmptyEqvAndEmpty LeftChopEqvChop*)
have *6*: $\vdash$ $(f \wedge empty);\#True = init(f)$
by (*simp add*: *AndChopCommute init-d-def*)
from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *WRFinEqvInit*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*fin f*) = *init* (*f*)
by (*metis EqvWReverseWReverse EqvWReverseWReverseAlt FiniteTransEqv RFinEqvInit WRevRev inteq-reflection*)

lemma *RFinEqvInitAlt*:
$\vdash$ $(\mathit{fin}\ f)^r = (\mathit{finite} \wedge \mathit{init}\ f\ )$
proof −
have *1*: $\vdash \mathit{fin}\ f = \#\mathit{True};(f \wedge \mathit{empty})$
using *FinEqvTrueChopAndEmpty* **by** *auto*
have *2*: $\vdash (\mathit{fin}\ f)^r = (\#\mathit{True};(f \wedge \mathit{empty}))^r$
using *1 REqvRule* **by** *blast*
have *3*: $\vdash (\#\mathit{True};(f \wedge \mathit{empty}))^r = (f \wedge \mathit{empty})^r;\mathit{finite}$
by (*metis RTrueAlt RevChopAlt inteq-reflection*)
have *4*: $\vdash (f\wedge\ \mathit{empty})^r;\mathit{finite} = (f^r \wedge \mathit{empty});\mathit{finite}$
using *LeftChopEqvChop RAnd REmptyEqvEmpty* **by** (*metis int-eq*)
have *5*: $\vdash (f\wedge\ \mathit{empty})^r;\mathit{finite} = (f\wedge\ \mathit{empty});\mathit{finite}$
by (*simp add*: *RAndEmptyEqvAndEmpty LeftChopEqvChop*)
have *6*: $\vdash (f\wedge\ \mathit{empty});\mathit{finite} = (\mathit{finite} \wedge \mathit{init}\ f\ )$
by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop lift-and-com*)

from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *WRFinEqvInitAlt*:
$\vdash$ $\mathit{wrev}\ (\mathit{fin}\ f) = (\mathit{finite} \longrightarrow \mathit{init}\ f)$
by (*metis EqvWReverseWReverseAlt FiniteImpAnd WRInitEqvFin all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RHaltEqvInitonly*:
$\vdash \mathit{finite} \longrightarrow (\mathit{halt}\ f)^r = \mathit{initonly}\ (f^r)$
proof −
have *1*: $\vdash (\mathit{halt}\ f)^r = (\Box\ (\ \mathit{empty} = f\ ))^r$ **by** (*simp add*: *halt-d-def*)
have *2*: $\vdash \mathit{finite} \longrightarrow (\Box\ (\ \mathit{empty} = f\ ))^r = \mathit{bi}\ (\ (\mathit{empty} = f)^r)$ **by** (*simp add*: *RBoxEqvBi*)
have *3*: $\vdash \mathit{finite} \longrightarrow (\mathit{empty} = f)^r = (\mathit{empty} = f^r)$ **by** (*metis REmptyEqvEmpty inteq-reflection rev-fun2*)
hence *4*: $\vdash \mathit{finite} \longrightarrow \mathit{bi}\ (\ (\mathit{empty} = f)^r) = \mathit{bi}(\mathit{empty} = f^r)$
by (*metis* (*no-types*, *opaque-lifting*) *FiniteChopEqv FiniteNotEqv bi-d-def di-d-def*)
have *5*: $\vdash \mathit{bi}(\mathit{empty} = f^r) = \mathit{initonly}(f^r)$ **by** (*simp add*: *initonly-d-def*)
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *WRHaltEqvInitonly*:
$\vdash \mathit{finite} \longrightarrow \mathit{wrev}\ (\mathit{halt}\ f) = \mathit{initonly}\ (\mathit{wrev}\ f)$
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverse FiniteImpAnd FiniteInitonlyEqv RHaltEqvInitonly WRevAndFiniteAlt WReverseEqvAlt all-wrev-eq*(*2*) *all-wrev-eq*(*9*) *int-eq*)

lemma *RHaltEqvInitonlyAlt*:
$\vdash$ $(\mathit{halt}\ f)^r = (\mathit{initonly}\ (f^r) \wedge \mathit{finite})$
proof −
have *1*: $\vdash (\mathit{halt}\ f)^r = (\Box\ (\ \mathit{empty} = f\ ))^r$ **by** (*simp add*: *halt-d-def*)
have *2*: $\vdash$ $(\Box\ (\ \mathit{empty} = f\ ))^r = (\mathit{bi}\ (\ (\mathit{empty} = f)^r) \wedge \mathit{finite})$ **by** (*simp add*: *RBoxEqvBiAlt*)
have *3*: $\vdash$ $(\mathit{empty} = f)^r = ((\mathit{empty} = f^r) \wedge \mathit{finite})$
using *REmptyEqvEmpty FiniteImpAnd*[*of LIFT* $(\mathit{empty} = f)^r$]
by (*metis* (*no-types*, *opaque-lifting*) *RAnd RTrueAlt int-simps*(*17*) *inteq-reflection lift-and-com rev-fun2*)
hence *4*: $\vdash$ $(\mathit{bi}\ (\ (\mathit{empty} = f)^r) \wedge \mathit{finite}) = (\mathit{bi}(\ (\mathit{empty} = f^r) \wedge \mathit{finite}) \wedge \mathit{finite})$
by (*metis 2 inteq-reflection*)

have *39*: $\vdash$ *bi finite = finite*
by (*metis AndInfChopEqvAndInf InfEqvNotFinite bi-d-def di-d-def finite-d-def int-simps*(*17*) *inteq-reflection*)

have *40*: $\vdash$ (*bi*((*empty* = f^r) $\wedge$ *finite*) $\wedge$ *finite*) = (*bi*((*empty* = f^r)) $\wedge$ *finite*)
by (*metis BiAndEqv BiElim 39 Prop10 Prop12 inteq-reflection*)
have *5*: $\vdash$ (*bi*(*empty* = f^r) $\wedge$ *finite*) = (*initonly*(f^r) $\wedge$ *finite*) **by** (*simp add*: *initonly-d-def*)
from *1 2 4 5* **show** *?thesis* **by** (*metis 40 inteq-reflection*)
qed

lemma *WRHaltEqvInitonlyAlt*:
$\vdash$ *wrev* (*halt f*) = (*finite* $\longrightarrow$ *initonly* (*wrev f*))
using *WRHaltEqvInitonly WRevRev* **by** *fastforce*

lemma *RInitonlyEqvHalt*:
$\vdash$ *finite* $\longrightarrow$ (*initonly f*)^r = *halt*(f^r)
proof −
have *1*: $\vdash$ (*initonly f*)^r = (*bi* (*empty* = *f*))^r **by** (*simp add*: *initonly-d-def*)
have *2*: $\vdash$ *finite* $\longrightarrow$ (*bi* (*empty* = *f*))^r = $\Box$((*empty* = *f*)^r) **by** (*simp add*: *RBiEqvBox*)
have *3*: $\vdash$ *finite* $\longrightarrow$ (*empty* = *f*)^r = (*empty* = f^r) **by** (*metis REmptyEqvEmpty inteq-reflection rev-fun2*)
hence *4*: $\vdash$ *finite* $\longrightarrow\Box$ ((*empty* = *f*)^r) = $\Box$(*empty* = f^r)
by (*metis* (*no-types*, *opaque-lifting*) *FiniteNotEqv FiniteRightChopEqvChop always-d-def sometimes-d-def*)
have *5*: $\vdash$ $\Box$(*empty* = f^r) = *halt*(f^r) **by** (*simp add*: *halt-d-def*)
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *WRInitonlyEqvHalt*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*initonly f*) = *halt*(*wrev f*)
by (*metis* (*no-types*, *lifting*) *RInitonlyEqvHalt WRHaltEqvInitonlyAlt WRevAndFiniteAlt WReverseEqvAlt*

all-wrev-eq(*2*) *all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RInitonlyEqvHaltAlt*:
$\vdash$ (*initonly f*)^r = (*halt*(f^r) $\wedge$ *finite*)
by (*metis* (*no-types*, *lifting*) *EqvReverseReverseAlt FiniteImpAnd RInitonlyEqvHalt RRAndAlt*
WRevAndFiniteAlt all-rev-eq(*10*) *inteq-reflection*)

lemma *WRInitonlyEqvHaltAlt*:
$\vdash$ *wrev* (*initonly f*) = (*finite* $\longrightarrow$ *halt*(*wrev f*))
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt RInitonlyEqvHaltAlt WRHaltEqvInitonlyAlt*
WRevAndFiniteAlt WRevRev all-wrev-eq(*9*) *inteq-reflection*)

lemma *RRHaltEqvInitonly*:
$\vdash$ *finite* $\longrightarrow$ (*halt* (f^r))^r = *initonly* (*f*)
proof −

have *1*: $\vdash$ *finite* $\longrightarrow$ $(halt\ (f^r))^r$ = *initonly* $((f^r)^r)$ **using** *RHaltEqvInitonly* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$ *initonly* $((f^r)^r)$ = *initonly*(f)
by (*simp add*: *EqvReverseReverse FiniteInitonlyEqv*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRHaltEqvInitonly*:
$\vdash$ *finite* $\longrightarrow$ *wrev* (*halt* (*wrev f*)) = *initonly* (*f*)
by (*meson EqvWReverseWReverse FiniteInitonlyEqv FiniteTransEqv WRHaltEqvInitonly*)

lemma *RRHaltEqvInitonlyAlt*:
$\vdash$ $(halt\ (f^r))^r$ = (*initonly* (*f*) $\wedge$ *finite*)
by (*metis EqvReverseReverse FiniteImpAnd FiniteInitonlyEqv RHaltEqvInitonlyAlt inteq-reflection*)

lemma *WRWRHaltEqvInitonlyAlt*:
$\vdash$ *wrev* (*halt* (*wrev f*)) = (*finite* $\longrightarrow$ *initonly* (*f*))
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt WRInitonlyEqvHaltAlt WRevAndFiniteAlt WRevRev all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RRInitonlyEqvHalt* :
$\vdash$ *finite* $\longrightarrow$ $(initonly\ (f^r))^r$ = *halt*(f)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ $(initonly\ (f^r))^r$ = *halt*$((f^r)^r)$ **using** *RInitonlyEqvHalt* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$ *halt*$((f^r)^r)$ = *halt*(f)
by (*simp add*: *EqvReverseReverse FiniteHaltEqv*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRInitonlyEqvHalt* :
$\vdash$ *finite* $\longrightarrow$ *wrev* (*initonly* (*wrev f*)) = *halt*(f)
by (*meson EqvWReverseWReverse FiniteHaltEqv FiniteTransEqv WRInitonlyEqvHalt*)

lemma *RRInitonlyEqvHaltAlt* :
$\vdash$ $(initonly\ (f^r))^r$ = (*halt*(f) $\wedge$ *finite*)
by (*metis EqvReverseReverse FiniteHaltEqv FiniteImpAnd RInitonlyEqvHaltAlt inteq-reflection*)

lemma *WRWRInitonlyEqvHaltAlt* :
$\vdash$ *wrev* (*initonly* (*wrev f*)) = (*finite* $\longrightarrow$ *halt*(f))
by (*metis EqvWReverseWReverseAlt WRHaltEqvInitonlyAlt WRInitonlyEqvHaltAlt WRWRHaltEqvInitonlyAlt*
inteq-reflection)

lemma *RKeepEqvKeep* :
$\vdash$ *finite* $\longrightarrow$ $(keep\ f)^r$ = *keep*(f^r)
proof −
have *1*: $\vdash$ $(keep\ f)^r$ = $(ba(skip \longrightarrow f))^r$ **by** (*simp add*: *keep-d-def*)
have *2*: $\vdash$ *finite* $\longrightarrow$ $(ba(skip \longrightarrow f))^r$ = $ba((skip \longrightarrow f)^r)$ **by** (*simp add*:*RBaEqvBa*)
have *3*: $\vdash$ *finite* $\longrightarrow$ $(skip \longrightarrow f)^r$ = $(skip \longrightarrow f^r)$
by (*metis all-rev-eq*(*8*) *rev-fun2*)

hence *4*: $\vdash$ *finite* $\longrightarrow$ *ba*((*skip* $\longrightarrow$ *f*)^r) = *ba*(*skip* $\longrightarrow$ *f*^{r})
by (*simp add*: *FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop ba-d-def da-d-def*)
have *5*: $\vdash$ *ba*(*skip* $\longrightarrow$ *f*^{r}) = *keep*(*f*^{r}) **by** (*simp add*: *keep-d-def*)
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *WRKeepEqvKeep* :
$\vdash$ *finite* $\longrightarrow$ *wrev* (*keep f*) = *keep*(*wrev f*)
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt FiniteImpAnd RKeepEqvKeep WRevAndFiniteAlt*

WReverseEqvAlt all-wrev-eq(*2*) *all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RKeepEqvKeepAlt* :
$\vdash$ (*keep f*)^r = (*keep*(*f*^{r}) $\wedge$ *finite*)
using *FiniteImpAnd*[*of LIFT* (*keep f*)^r] *RKeepEqvKeep*
by (*metis* (*no-types*, *opaque-lifting*) *Prop11 Prop12 RImpRule RTrueAlt int-simps*(*12*) *int-simps*(*13*)
inteq-reflection)

lemma *WRKeepEqvKeepAlt* :
$\vdash$ *wrev* (*keep f*) = (*finite* $\longrightarrow$ *keep*(*wrev f*))
using *WRKeepEqvKeep WRevRev* **by** *fastforce*

lemma *RRKeepEqvKeep* :
$\vdash$ *finite* $\longrightarrow$ (*keep* (*f*^{r}))^r = *keep*(*f*)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ (*keep* (*f*^{r}))^r = *keep*((*f*^{r})^r) **using** *RKeepEqvKeep* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow$ *keep*((*f*^{r})^r) = *keep*(*f*)
by (*simp add*: *EqvReverseReverse FiniteKeepEqv*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRKeepEqvKeep* :
$\vdash$ *finite* $\longrightarrow$ *wrev* (*keep* (*wrev f*)) = *keep*(*f*)
by (*meson EqvWReverseWReverse FiniteKeepEqv FiniteTransEqv WRKeepEqvKeep*)

lemma *RRKeepEqvKeepAlt* :
$\vdash$ (*keep* (*f*^{r}))^r = (*keep*(*f*) $\wedge$ *finite*)
by (*metis EqvReverseReverse FiniteImpAnd FiniteKeepEqv RKeepEqvKeepAlt inteq-reflection*)

lemma *WRWRKeepEqvKeepAlt* :
$\vdash$ *wrev* (*keep* (*wrev f*)) = (*finite* $\longrightarrow$ *keep*(*f*))
by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt WRKeepEqvKeepAlt WRevAndFiniteAlt WRevRev*

all-wrev-eq(*9*) *inteq-reflection*)

lemma *RassignEqvTAssign*:

$\vdash$ *finite* $\longrightarrow$ $(\$v = e)^R = (v \leftarrow e^R)$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ $(\$v = e)^R = ((\$v)^R = e^R)$
using *rev-fun2-e* **by** *blast*
have *20*: $\vdash$ *finite* $\longrightarrow$ $(\$v)^R = (!v)$
by (*simp add*: *rev-current*)
have *2*: $\vdash$ *finite* $\longrightarrow$ $((\$v)^R = e^R) = ((!v) = e^R)$
using *20* **by** (*auto simp add*: *Valid-def itl-defs*)
have *3*: $\vdash$ *finite* $\longrightarrow$ $((!v) = e^R) = (v \leftarrow e^R)$ **by** (*simp add*: *intI temporal-assign-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *RassignEqvTAssignAlt*:
$\vdash$ $((\$v = e)^R \wedge$ *finite*$) = ((v \leftarrow e^R) \wedge$ *finite*$)$
by (*simp add*: *FiniteImpAnd RassignEqvTAssign*)

lemma *RTAssignEqvAssign*:
$\vdash$ *finite* $\longrightarrow$ $(v \leftarrow e)^r = (\$v = e^R)$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ $(v \leftarrow e)^r =$ (*finite* $\longrightarrow$ $!v = e)^r$
unfolding *temporal-assign-d-def*
by (*metis int-simps*(*1*) *int-simps*(*12*) *inteq-reflection*)
have *2*: $\vdash$ *finite* $\longrightarrow$ (*finite* $\longrightarrow$ $!v = e)^r = (\$v = e^R)$
by (*auto simp add*: *Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev nlast-nrev*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *WRTAssignEqvAssign*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(v \leftarrow e) = (\$v = e^R)$
by (*metis* (*no-types*, *opaque-lifting*) *FiniteNotEqv FiniteTransEqv RTAssignEqvAssign int-simps*(*4*)
inteq-reflection rev-fun1 wfreverse-d-def)

lemma *RTAssignEqvAssignAlt*:
$\vdash$ $(v \leftarrow e)^r = ((\$v = e^R) \wedge$ *finite*$)$
by (*metis FiniteImpAnd RAnd RTAssignEqvAssign RTrueAlt int-simps*(*17*) *inteq-reflection lift-and-com*)

lemma *WRTAssignEqvAssignAlt*:
$\vdash$ *wrev* $(v \leftarrow e) =$ (*finite* $\longrightarrow$ $(\$v = e^R)$)
using *RTAssignEqvAssignAlt WRevRev* **by** *fastforce*

lemma *RNextAssignEqvPrevAssign*:
$\vdash$ *finite* $\longrightarrow$ $(v := e)^r = (v =: e^R)$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ $(v := e)^r = (v\$ = e)^r$ **by** (*simp add*: *REqvRule intI next-assign-d-def*)
have *20*: $\vdash$ *finite* $\longrightarrow$ $(v\$ = e)^r = ((v\$)^R = e^R)$
by (*simp add*: *ereverse-d-def freverse-d-def intI itl-defs*(*5*))
have *21*: $\vdash$ *finite* $\longrightarrow$ $((v\$)^R = e^R) = (v! = e^R)$
by (*simp add*: *Valid-def itl-defs freverse-d-def ereverse-d-def*)
(*metis One-nat-def Suc-ile-eq epred-conv-minus gr-zeroI idiff-enat-enat nfinite-nlength-enat*

nnth-nrev one-enat-def the-enat.simps zero-enat-def)
have *2*: $\vdash$ *finite* $\longrightarrow$ $(v\$ = e)^r = (v! = e^R)$
using *20 21 FiniteTransEqv* **by** *blast*
have *3*: $\vdash$ *finite* $\longrightarrow$ $(v! = e^R) = (v =: e^R)$
by (*auto simp add*: *Valid-def itl-defs freverse-d-def ereverse-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *WRNextAssignEqvPrevAssign*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(v := e) = (v =: e^R)$
by (*metis EqvWReverseWReverse EqvWReverseWReverseAlt FiniteTransEqv RNextAssignEqvPrevAssign WRevRev*
inteq-reflection)

lemma *RNextAssignEqvPrevAssignAlt*:
$\vdash$ $(v := e)^r = ((v =: e^R) \wedge$ *finite*$)$
using *FiniteImpAnd*[*of LIFT* $(v := e)^r$] *RNextAssignEqvPrevAssign*[*of v e*]
by (*metis* (*no-types*, *opaque-lifting*) *Prop11 Prop12 RImpRule RTrueAlt int-simps*(*12*) *int-simps*(*13*)
inteq-reflection)

lemma *WRNextAssignEqvPrevAssignAlt*:
$\vdash$ *wrev* $(v := e) =$ (*finite* $\longrightarrow$ $(v =: e^R)$)
by (*metis EqvWReverseWReverseAlt RNextAssignEqvPrevAssignAlt WRevRev all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RPrevAssignEqvNextAssign*:
$\vdash$ *finite* $\longrightarrow$ $(v =: e)^r = (v := e^R)$
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ $(v =: e)^r = (v! = e)^r$
by (*auto simp add*: *Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev*)
have *2*: $\vdash$ *finite* $\longrightarrow$ $(v! = e)^r = (v\$ = e^R)$
by (*simp add*: *Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev*)
(*metis One-nat-def Suc-ile-eq epred-enat i0-less ndropn-nfirst nfinite-ndropn nfinite-nlength-enat*
nlast-nrev nrev-ndropn ntaken-nlast the-enat.simps zero-enat-def)
have *3*: $\vdash$ *finite* $\longrightarrow$ $(v\$ = e^R) = (v := e^R)$ **by** (*simp add*: *next-assign-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *WRPrevAssignEqvNextAssign*:
$\vdash$ *finite* $\longrightarrow$ *wrev* $(v =: e) = (v := e^R)$
by (*metis* (*no-types*, *opaque-lifting*) *FiniteNotEqv FiniteTransEqv RPrevAssignEqvNextAssign int-simps*(*4*)

inteq-reflection rev-fun1 wfreverse-d-def)

lemma *RPrevAssignEqvNextAssignAlt*:
$\vdash$ $(v =: e)^r = ((v := e^R) \wedge$ *finite*$)$
using *FiniteImpAnd*[*of LIFT* $(v =: e)^r$] *RPrevAssignEqvNextAssign*
by (*metis* (*no-types*, *opaque-lifting*) *Prop11 Prop12 RImpRule RTrueAlt int-simps*(*12*)
int-simps(*13*) *inteq-reflection*)

lemma *WRPrevAssignEqvNextAssignAlt*:

$\vdash$ *wrev* $(v =: e) = (\mathit{finite} \longrightarrow (v := e^R))$
by (*metis EqvWReverseWReverseAlt RPrevAssignEqvNextAssignAlt WRevRev all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RGetsEqvBaSkipImp*:
$\vdash \mathit{finite} \longrightarrow (v \ \mathit{gets}\ e)^r = \mathit{ba}(\mathit{skip} \longrightarrow (\$v = e^R))$
proof −
have *01*: $\vdash (\mathit{skip} \longrightarrow \mathit{finite} \longrightarrow\ !v = e) =$
$(\mathit{skip}\ \longrightarrow\ !v = e)$
using *nlength-eq-enat-nfiniteD* **by** (*simp add*: *Valid-def itl-defs*, *blast*)
have *1*: $\vdash \mathit{finite} \longrightarrow (v \ \mathit{gets}\ e)^r = (\mathit{ba}(\mathit{skip} \longrightarrow (!v = e)))^r$
unfolding *gets-d-def keep-d-def temporal-assign-d-def*
by (*metis 01 EqvReverseReverse REmptyEqvEmpty int-simps*(*1*) *inteq-reflection*)
have *2*: $\vdash \mathit{finite} \longrightarrow (\mathit{ba}(\mathit{skip} \longrightarrow (!v = e)))^r = \mathit{ba}\ (\ (\mathit{skip} \longrightarrow (!v = e))^r\ )$
by (*simp add*: *RBaEqvBa*)
have *3*: $\vdash \mathit{finite} \longrightarrow (\mathit{skip} \longrightarrow (!v = e))^r = (\mathit{skip} \longrightarrow (\$v = e^R))$
by (*auto simp add*: *Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev nlast-nrev*)
hence *4*: $\vdash \mathit{finite} \longrightarrow \mathit{ba}\ ((\mathit{skip} \longrightarrow (!v = e))^r) = \mathit{ba}\ (\mathit{skip} \longrightarrow (\$v = e^R))$
by (*simp add*: *FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop ba-d-def da-d-def*)
from *1 2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *WRGetsEqvBaSkipImp*:
$\vdash \mathit{finite} \longrightarrow \mathit{wrev}\ (v \ \mathit{gets}\ e) = \mathit{ba}(\mathit{skip} \longrightarrow (\$v = e^R))$
by (*metis FiniteKeepEqv FiniteTransEqv WRKeepEqvKeep WRTAssignEqvAssign gets-d-def keep-d-def*)

lemma *RGetsEqvBaSkipImpAlt*:
$\vdash\ (v \ \mathit{gets}\ e)^r = (\mathit{ba}(\mathit{skip} \longrightarrow (\$v = e^R)) \wedge \mathit{finite})$
by (*metis FiniteImpAnd RAnd RGetsEqvBaSkipImp RTrueAlt int-simps*(*17*) *inteq-reflection lift-and-com*)

lemma *WRGetsEqvBaSkipImpAlt*:
$\vdash$ *wrev* $(v \ \mathit{gets}\ e) = (\mathit{finite} \longrightarrow \mathit{ba}(\mathit{skip} \longrightarrow (\$v = e^R)))$
by (*metis EqvWReverseWReverseAlt RGetsEqvBaSkipImpAlt WRevRev all-wrev-eq*(*9*) *inteq-reflection*)

lemma *RIfThenElse*:
$\vdash \mathit{finite} \longrightarrow (\mathit{if}_i\ \mathit{f0}\ \mathit{then}\ \mathit{f1}\ \mathit{else}\ \mathit{f2})^r = \mathit{if}_i\ (\mathit{f0}^r)\ \mathit{then}\ (\mathit{f1}^r)\ \mathit{else}\ (\mathit{f2}^r)$
unfolding *ifthenelse-d-def*
by (*auto simp add*: *Valid-def itl-defs freverse-d-def ereverse-d-def*)

lemma *WRIfThenElse*:
$\vdash \mathit{finite} \longrightarrow \mathit{wrev}\ (\mathit{if}_i\ \mathit{f0}\ \mathit{then}\ \mathit{f1}\ \mathit{else}\ \mathit{f2}) = \mathit{if}_i\ (\mathit{wrev}\ \mathit{f0})\ \mathit{then}\ (\mathit{wrev}\ \mathit{f1})\ \mathit{else}\ (\mathit{wrev}\ \mathit{f2})$
unfolding *ifthenelse-d-def*
by (*metis* (*no-types*, *lifting*) *FiniteAndEqv FiniteOrEqv WRAnd WRNot WROrAlt int-simps*(*1*)
inteq-reflection wrev-fun2)

lemma *RIfThenElseAlt*:
$\vdash (\mathit{if}_i\ \mathit{f0}\ \mathit{then}\ \mathit{f1}\ \mathit{else}\ \mathit{f2})^r = (\mathit{if}_i\ (\mathit{f0}^r)\ \mathit{then}\ (\mathit{f1}^r)\ \mathit{else}\ (\mathit{f2}^r) \wedge \mathit{finite})$
using *RIfThenElse*[*of f0 f1 f2*]

FiniteImpAnd[*of LIFT* $(if_i\ f0\ then\ f1\ else\ f2)^r$]
by (*metis RAnd RTrueAlt int-simps*(17) *inteq-reflection lift-and-com*)

lemma *WRIfThenElseAlt*:
$\vdash$ *wrev* $(if_i\ f0\ then\ f1\ else\ f2) = (finite \longrightarrow if_i\ (wrev\ f0)\ then\ (wrev\ f1)\ else\ (wrev\ f2)\ )$
by (*metis* (*no-types*, *lifting*) *FiniteImp FiniteImpAnd Prop12 WRIfThenElse WRevAndFiniteAlt WRevFinite WRevRev int-iffD1 int-iffI inteq-reflection*)

lemma *RWhile*:
$\vdash (init\ f \wedge swhile\ f0\ do\ f1)^r = (\ fin(f) \wedge (schopstar\ ((f0^r) \wedge (f1^r))) \wedge init\ (\neg(f0))\ )$
proof −
have *1*: $\vdash (init\ f \wedge swhile\ f0\ do\ f1)^r = (\ init\ f \wedge (schopstar\ (f0 \wedge f1)) \wedge sfin\ (\neg f0)\ )^r$
by (*simp add*: *swhile-d-def*)
have *2*: $\vdash (\ init\ f \wedge (schopstar\ (f0 \wedge f1)) \wedge sfin\ (\neg f0)\ )^r =$
$((init\ f)^r \wedge (schopstar\ (f0 \wedge f1))^r \wedge (sfin\ (\neg f0))^r)$
by (*metis RAnd inteq-reflection*)
have *3*: $\vdash finite \longrightarrow (init\ f)^r = fin(f)$
by (*simp add*: *RInitEqvFin*)
have *4*: $\vdash (schopstar\ (f0 \wedge f1))^r = (schopstar\ ((f0^r) \wedge (f1^r)))$
by (*metis RAnd RevChopstar inteq-reflection*)
have *40*: $\vdash finite \longrightarrow (sfin\ (\neg f0)) = (fin\ (\neg f0))$
by (*metis* (*no-types*, *opaque-lifting*) *FinEqvTrueChopAndEmpty Finprop*(4) *int-simps*(4) *inteq-reflection sfin-d-def*)
have *41*: $\vdash finite \longrightarrow (sfin\ (\neg f0))^r = (fin\ (\neg f0))^r$
by (*metis 40 Prop12 ReverseEqv all-rev-eq*(2))
have *5*: $\vdash finite \longrightarrow (sfin\ (\neg f0))^r = init\ (\neg(f0))$
using *41 FiniteTransEqv RFinEqvInit* **by** *blast*
have *50*: $\vdash (schopstar\ (f0 \wedge f1))^r \longrightarrow finite$
using *4 SChopstar-finite* **by** *fastforce*
have *51*: $\vdash (schopstar\ ((f0^r) \wedge (f1^r))) \longrightarrow finite$
using *SChopstar-finite* **by** *blast*
have *6*: $\vdash ((init\ f)^r \wedge (schopstar\ (f0 \wedge f1))^r \wedge (sfin\ (\neg f0))^r) =$
$(\ fin(f) \wedge (schopstar\ ((f0^r) \wedge (f1^r))) \wedge init\ (\neg(f0))\ )$
using *3 4 5 50 51* **by** *fastforce*
from *1 2 6* **show** *?thesis* **by** *fastforce*
qed

lemma *WRWhile*:
$\vdash wrev\ (init\ f \wedge swhile\ f0\ do\ f1) =$
$(finite \longrightarrow fin(f) \wedge (schopstar\ ((wrev\ f0) \wedge (wrev\ f1))) \wedge init\ (\neg(f0))\ )$
by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt RAnd RWhile SChopstar-Chopstar WRevAndFiniteAlt WRevRev all-wrev-eq*(9) *int-eq*)

lemma *AAxRev*:
$\vdash (\forall\forall\ x.\ F\ x)^r = (\forall\forall\ x.\ (F\ x)^r)$
proof −

have *1*: $\vdash (\forall\forall\ x.\ F\ x) = (\neg(\exists\exists\ x.\ \neg(F\ x)))$ **using** *AAxDef* **by** *blast*
have *2*: $\vdash (\forall\forall\ x.\ F\ x)^r = (\neg(\exists\exists\ x.\ \neg(F\ x)))^r$ **using** *REqvRule 1* **by** *blast*
have *3*: $\vdash\ (\neg(\exists\exists\ x.\ \neg(F\ x)))^r = (\neg((\exists\exists\ x.(\neg\ (F\ x)))^r) \wedge finite)$
using *FiniteImpAnd*[*of LIFT* $(\neg(\exists\exists\ x.\ \neg(F\ x)))^r$]
by (*simp add*: *rev-fun1-alt*)
have *4*: $\vdash ((\exists\exists\ x.(\neg\ (F\ x)))^r) = ((\exists\exists\ x.(\neg\ (F\ x))^r))$ **by** (*simp add*: *EExRev*)
hence *5*: $\vdash (\neg((\exists\exists\ x.(\neg\ (F\ x)))^r) \wedge finite) = (\neg(\exists\exists\ x.(\neg\ (F\ x))^r) \wedge finite)$ **by** *auto*
have *51*: $\bigwedge x. \vdash\ ((\neg\ (F\ x))^r) = ((\neg(\ (F\ x)^r) \wedge finite))$
using *rev-fun1-alt* **by** *blast*
hence *52*: $\vdash\ (\exists\exists\ x.((\neg\ (F\ x))^r)) = (\exists\exists\ x.(\neg(\ (F\ x)^r) \wedge finite))$
using *EExEqvRule*[*of* $(\lambda x.\ LIFT(\ (\neg\ (F\ x)))^r\ )\ (\lambda x.\ LIFT((\neg(\ (F\ x)^r)) \wedge finite))\ ]$
by *fastforce*
hence *6*: $\vdash (\neg(\exists\exists\ x.(\neg\ (F\ x))^r) \wedge finite\ ) = (\neg(\exists\exists\ x.(\neg(\ (F\ x)^r) \wedge finite)) \wedge finite)$
by *auto*
have *60*: $\vdash (\neg(\exists\exists\ x.(\neg(\ (F\ x)^r) \wedge finite)) \wedge finite) =$
$(\neg(\exists\exists\ x.\ (\neg\ ((\ (F\ x)^r) \vee \neg finite))) \wedge finite)$
by (*auto simp add*: *Valid-def exist-state-d-def*)
have *7*: $\vdash (\neg(\exists\exists\ x.(\neg((\ (F\ x)^r) \vee \neg finite))\ ) \wedge finite\ \ ) = ((\forall\forall\ x.\ ((F\ x)^r \vee \neg finite)) \wedge finite)$ **using** *AAxDef*
by (*simp add*: *forall-state-d-def*)
have *69*: $\vdash ((\forall\forall\ x.\ ((F\ x)^r \vee \neg finite)) \wedge finite) = (\forall\forall\ x.\ (finite \wedge ((F\ x)^r \vee \neg finite)))$
using *AAxAndImport*[*of LIFT finite* $\lambda x.\ LIFT\ (F\ x)^r \vee \neg finite\ \ ]$ **by** *auto*
have *70*: $\bigwedge x. \vdash\ (finite \wedge ((F\ x)^r \vee \neg finite)) =\ (F\ x)^r$
by (*auto simp add*: *freverse-d-def itl-defs*(*5*))
have *71*: $\vdash ((\forall\forall\ x.\ (finite \wedge ((F\ x)^r \vee \neg finite)))) =\ (\forall\forall\ x.\ (F\ x)^r)$
by (*simp add*: *70 AAxEqvRule*)
from *1 2 3 5 6 7 70* **show** *?thesis*
by (*metis 60 69 71 inteq-reflection*)
qed

lemma *AAxWRev*:
$\vdash$ *wrev* $(\forall\forall\ x.\ F\ x) = (\forall\forall\ x.\ \ wrev\ (F\ x))$
proof −
have *1*: $\vdash$ *wrev* $(\forall\forall\ x.\ F\ x) = (finite \longrightarrow frev\ (\forall\forall\ x.\ F\ x))$
by (*simp add*: *WRevRev*)
have *2*: $\vdash (finite \longrightarrow frev\ (\forall\forall\ x.\ F\ x)) = (finite \longrightarrow\ (\forall\forall\ x.\ frev\ (F\ x)))$
using *AAxRev*[*of F*] **by** *auto*
have *3*: $\vdash (finite \longrightarrow\ (\forall\forall\ x.\ frev\ (F\ x))) = (\forall\forall\ x.\ \neg\ finite \vee frev\ (F\ x))$
using *AAxOrImport*[*of LIFT* $\neg finite\ \lambda x.\ LIFT\ frev\ (F\ x)$] **by** *auto*
have *4*: $\bigwedge x. \vdash (\neg\ finite \vee frev\ (F\ x)) = (wrev\ (F\ x))$
using *WRevRev* **by** *fastforce*
have *5*: $\vdash (\forall\forall\ x.\ \neg\ finite \vee frev\ (F\ x)) = (\forall\forall\ x.\ \ wrev\ (F\ x))$
by (*meson 4 AAxEqvRule*)
show *?thesis*
by (*metis 1 2 3 5 inteq-reflection*)
qed

end

3.13 First occurrence operator

theory *First*
imports
Omega
begin

Runtime verification (RV) has gained significant interest in recent years. The behaviour of a program can be verified in real time by analysing its evolving trace. This approach has two significant benefits over static verification techniques such as model checking. Firstly, it is only necessary to verify actual execution paths rather than all possible paths. Secondly, it is possible to react at runtime should the program diverge from its specified behaviour. RV does not replace traditional verification techniques but it does provide an extra layer of security.

Linear Temporal Logic (LTL) is a popular formalism for writing specifications from which RV monitors can be derived automatically. By contrast, Interval Temporal Logic (ITL) has not been as widely represented in this field despite being more expressive and compositional. The principal issue is efficiency. ITL uses non-deterministic operators to IConstruct sequential and iterative specifications (chop and chop-star, respectively) and these introduce combinatorial complexity. Approaches to mitigate this include using a deterministic subset of ITL or adapting the semantics to include a deterministic chop operator. This work proposes an alternative approach, wholly within existing ITL, and based upon a new, derived operator called "first occurrence".

A theory of first occurrence is developed and used to derive an algebra of RV monitors.

3.13.1 Definitions

Definitions Strict Initial and Final

definition *fprev-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *fprev-d F* ≡ *LIFT*(*F*⌢*skip*)

syntax
-fprev-d :: *lift* ⇒ *lift* ((*fprev* -) [*88*] *87*)

syntax (*ASCII*)
-fprev-d :: *lift* ⇒ *lift* ((*fprev* -) [*88*] *87*)

translations
-fprev-d ⇌ *CONST fprev-d*

definition *wfprev-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *wfprev-d F* ≡ *LIFT*(¬(*fprev*(¬*F*)))

syntax
-wfprev-d :: *lift* ⇒ *lift* ((*wfprev* -) [*88*] *87*)

syntax (*ASCII*)

-wfprev-d :: *lift* ⇒ *lift* ((*wfprev* -) [*88*] *87*)

translations
-wfprev-d ⇌ *CONST wfprev-d*

definition *bs-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
bs-d f ≡ *LIFT*(*empty* ∨ ((*bi f*) ⌢ *skip*))

syntax
-bs-d :: *lift* ⇒ *lift* ((*bs* -) [*88*] *87*)

syntax (*ASCII*)
-bs-d :: *lift* ⇒ *lift* ((*bs* -) [*88*] *87*)

translations
-bs-d ⇌ *CONST bs-d*

definition *ds-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
ds-d f ≡ *LIFT* (¬ (*bs* (¬ *f*)))

syntax
-ds-d :: *lift* ⇒ *lift* ((*ds* -) [*88*] *87*)

syntax (*ASCII*)
-ds-d :: *lift* ⇒ *lift* ((*ds* -) [*88*] *87*)

translations
-ds-d ⇌ *CONST ds-d*

Definition First and Last Operators

definition *first-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
first-d f ≡ *LIFT* (*f* ∧ (*bs* (¬ *f*)))

syntax
-first-d :: *lift* ⇒ *lift* ((▷ -) [*88*] *87*)

syntax (*ASCII*)
-first-d :: *lift* ⇒ *lift* ((*first* -) [*88*] *87*)

translations
-first-d ⇌ *CONST first-d*

3.13.2 Semantic Theorems

3.13.3 Bi induction

lemma *finite-ntaken*:
nfinite w $\Longrightarrow$ *f w* = *f* (*ntaken* (*the-enat* (*nlength w*)) *w*)
by (*metis ndropn-eq-NNil ndropn-nlast ntaken-all*)

lemma *biinduct-help*:
(∀ *n*. *enat n* ≤ *nlength w* ⟶ *f* (*ntaken n w*) ⟶ *n* = *0* ∨ *f* (*ntaken* (*n* − *Suc 0*) *w*)) $\Longrightarrow$
f w $\Longrightarrow$
nfinite w $\Longrightarrow$
enat j ≤ *nlength w* $\Longrightarrow$
enat k = *nlength w* $\Longrightarrow$
f (*ntaken j w*)
proof (*induct k arbitrary*: *w j*)
case *0*
then show *?case*
by (*metis le-zero-eq ntaken-all zero-enat-def*)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: *f* (*ntaken* (*k*+*1*) *w*)
by (*simp add*: *Suc.prems*(*2*) *Suc.prems*(*5*) *ntaken-all*)
have *2*: *f* (*ntaken* (*k*) *w*)
by (*metis 1 One-nat-def Suc.prems*(*1*) *Suc.prems*(*5*) *Suc-eq-plus1 Zero-not-Suc diff-Suc-1 nle-le*)
have *3*: *w* = *nappend* (*ntaken k w*) (*NNil* (*nlast*(*w*)))
by (*metis Suc.prems*(*5*) *nappend-ntaken-ndropn ndropn-all order-refl*)
have *4*: ∀ *n*. *enat n* ≤ *nlength* (*ntaken* (*k*) *w*) ⟶ *f* (*ntaken n* (*ntaken* (*k*) *w*)) ⟶
n = *0* ∨ *f* (*ntaken* (*n* − *Suc 0*) (*ntaken* (*k*) *w*))
by (*metis 3 Suc.prems*(*1*) *diff-le-self min.bounded-iff ntaken-eq-ntaken-antimono*
ntaken-nappend1 ntaken-nlength)
have *5*: *j* = *nlength w* $\Longrightarrow$ *?thesis*
by (*simp add*: *Suc.prems*(*2*) *ntaken-all*)
have *6*: *j*< *nlength w* $\Longrightarrow$ *?thesis*
using *2 3 4 Suc.hyps*[*of* (*ntaken k w*)]
by (*metis* (*no-types*, *lifting*) *Suc.prems*(*5*) *Suc-ile-eq linorder-not-less min.orderE*
nfinite-ntaken nle-le ntaken-nappend1 ntaken-nlength)
show *?thesis*
using *5 6 Suc.prems*(*4*) *order.order-iff-strict* **by** *blast*
qed
qed

lemma *BiInduct*:
⊢ *bi*(*f* ⟶ *wprev f*) ∧ *f* ∧ *finite* ⟶ *bi f*
proof −
have *1*: ⋀*j w*.

$(\forall\, n.\ enat\ n \leq nlength\ w \longrightarrow f\ (ntaken\ n\ w) \longrightarrow n = 0 \vee f\ (ntaken\ (n - Suc\ 0)\ w)) \Longrightarrow$
$f\ w \Longrightarrow$
$nfinite\ w \Longrightarrow$
$enat\ j \leq nlength\ w \Longrightarrow$
$f\ (ntaken\ j\ w)$
by (*metis biinduct-help nfinite-conv-nlength-enat*)
have *2*: $(\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge finite \longrightarrow bi\ f) =$
$(\forall\ w\ n.\ (\forall\, n.\ enat\ n \leq nlength\ w \longrightarrow f\ (ntaken\ n\ w) \longrightarrow n = 0 \vee f\ (ntaken\ (n - Suc\ 0)\ w)) \longrightarrow$
$f\ w \longrightarrow nfinite\ w \longrightarrow enat\ n \leq nlength\ w \longrightarrow f\ (ntaken\ n\ w))$
by (*auto simp add*: *Valid-def itl-defs*)

show *?thesis* **using** *1 2* **by** *blast*
qed

Semantics First Operator

lemma *FstAndBfsem*:
$(nlength\ \sigma > 0 \wedge (\sigma \models f) \wedge (\ \sigma \models\ \ bi\ (\neg f) \frown skip)) =$
$(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge (\forall\, ia{<}nlength\ (\sigma).\ (ntaken\ ia\ \ \sigma \models \neg f))\ )$
proof −
have $(nlength\ \sigma > 0 \wedge (\sigma \models f) \wedge (\ \sigma \models\ \ bi\ (\neg f) \frown skip)) =$
$(0 < nlength\ \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
$(\exists\, i.\ (i \leq nlength\ \sigma \longrightarrow (\forall\, ia{\leq}i.\ \neg\ (ntaken\ ia\ (ntaken\ i\ \sigma) \models f)) \wedge$
$nlength\ \sigma - i = Suc\ 0) \wedge i \leq nlength\ \sigma)$
)
unfolding *itl-defs* **by** *simp*
(*metis enat-ord-simps*(*1*) *ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD order-subst2*)
also have ... =
$(0 < nlength\ \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
$(\exists\, i.\ (i \leq nlength\ \sigma \longrightarrow (\forall\, ia{\leq}i.\ \neg\ (ntaken\ ia\ (ntaken\ i\ \sigma) \models f)) \wedge$
$i = nlength\ \sigma - Suc\ 0) \wedge i \leq nlength\ \sigma)$
)
by (*auto simp add*: *nfinite-conv-nlength-enat*)
(*metis Suc-diff-Suc cancel-comm-monoid-add-class.diff-cancel diff-zero lessI ntaken-all*
order.order-iff-strict zero-less-iff-neq-zero)
also have ... =
$(0 < nlength\ \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
$(\forall\, ia.\ enat\ ia{\leq}(nlength\ \sigma - Suc\ 0) \longrightarrow \neg\ (ntaken\ ia\ (ntaken\ (the\text{-}enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models f)\ )$
)
using *diff-le-self* **by** (*auto simp add*: *min-def nfinite-conv-nlength-enat*, *blast*)
also have ... =
$(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
$(\forall\, ia.\ ia{<}nlength\ (\ \sigma) \longrightarrow \neg\ (ntaken\ ia\ (ntaken\ (the\text{-}enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models f))$
)
by (*metis Suc-pred enat-ord-simps*(*1*) *enat-ord-simps*(*2*) *idiff-enat-enat less-Suc-eq-le*
nfinite-conv-nlength-enat zero-enat-def)
also have ... =
$(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
$(\forall\, ia.\ ia{<}nlength\ (\ \sigma) \longrightarrow (ntaken\ ia\ (ntaken\ (the\text{-}enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models \neg f))$

)
by *auto*
also have ... =
(*nlength* $\sigma > 0 \wedge$ *nfinite* $\sigma \wedge (\sigma \models f) \wedge (\forall$ *ia*<*nlength* (σ). (*ntaken ia* $\sigma \models \neg f)$))
by (*metis Suc-pred enat-ord-simps*(*2*) *idiff-enat-enat less-Suc-eq-le min-def*
nfinite-conv-nlength-enat ntaken-ntaken the-enat.simps zero-enat-def)
finally show (*nlength* $\sigma > 0 \wedge (\sigma \models f) \wedge ($ $\sigma \models$ *bi* $(\neg f) \frown$ *skip*)) =
(*nlength* $\sigma > 0 \wedge$ *nfinite* $\sigma \wedge (\sigma \models f) \wedge (\forall$ *ia*<*nlength* (σ). (*ntaken ia* $\sigma \models \neg f)$)) .
qed

lemma *Fstsem-0*:
$(\sigma \models \triangleright f) =$
(
($\sigma \models f \wedge$ *empty*) $\vee$ (*nlength* $\sigma > 0 \wedge (\sigma \models f) \wedge ($ $\sigma \models$ *bi* $(\neg f) \frown$ *skip*))
)
using *empty-defs* **by** (*auto simp add*: *first-d-def bs-d-def*)

lemma *Emptysem*:
$(\sigma \models f \wedge$ *empty*$) = ((\sigma \models f) \wedge$ *nlength* $\sigma = 0)$
using *empty-defs* **by** *auto*

lemma *Fstsem*:
$(\sigma \models \triangleright f) =$
(
($(\sigma \models f) \wedge$ *nlength* $\sigma = 0) \vee$
(*nlength* $\sigma > 0 \wedge$ *nfinite* $\sigma \wedge (\sigma \models f) \wedge (\forall$ *ia*<*nlength* (σ). (*ntaken ia* $\sigma \models \neg f)$))
)
using *Fstsem-0 Emptysem FstAndBfsem* **by** *metis*

Various Semantic Lemmas

lemma *DiLensem*:
$(\sigma \models$ *di* $(f \wedge$ *len*$(i)))) =$
((*ntaken i* $\sigma \models f) \wedge i \leq$*nlength* σ)
using *nlength-eq-enat-nfiniteD* **by** (*auto simp add*: *itl-defs len-defs*)

lemma *DfLensem*:
$(\sigma \models$ *df* $(f \wedge$ *len*$(i)))) =$
((*ntaken i* $\sigma \models f) \wedge i \leq$*nlength* σ)
by (*auto simp add*: *itl-defs len-defs*)

lemma *PrefixFstsem*:
((*ntaken i* $\sigma \models \triangleright f) \wedge i \leq$*nlength* σ) =
($i \leq$*nlength* $\sigma \wedge$
(
((*ntaken i* $\sigma \models f) \wedge i = 0) \vee$
($i > 0 \wedge$ (*ntaken i* $\sigma \models f) \wedge (\forall$ *ia*<*i*. (*ntaken ia* $\sigma \models \neg f)$))
)
)
proof −

have *1*: (((*ntaken i σ*) ⊨ ▷*f*)) =
 (
  (((*ntaken i σ*) ⊨ *f*) ∧ *nlength* (*ntaken i σ*) = *0*) ∨
  (*nlength* (*ntaken i σ*)>*0* ∧ ((*ntaken i σ*) ⊨ *f*) ∧
    (∀ *ia*<*nlength* (*ntaken i σ*). (*ntaken ia* (*ntaken i σ*) ⊨ ¬*f*)))
)
      using *Fstsem* **by** (*metis nfinite-ntaken*)
hence *2*: (((*ntaken i σ*) ⊨ ▷*f*) ∧ *i*≤*nlength σ*) =
 (*i*≤*nlength σ* ∧ (
  (((*ntaken i σ*) ⊨ *f*) ∧ *nlength* (*ntaken i σ*) = *0*) ∨
  (*nlength* (*ntaken i σ*)>*0* ∧ ((*ntaken i σ*) ⊨ *f*) ∧
    (∀ *ia*<*nlength* (*ntaken i σ*). (*ntaken ia* (*ntaken i σ*) ⊨ ¬*f*)))
)
)
      by *auto*
hence *3*: (((*ntaken i σ*) ⊨ ▷*f*) ∧ *i*≤*nlength σ*) =
 (*i*≤*nlength σ* ∧ (
  (((*ntaken i σ*) ⊨ *f*) ∧ *i* = *0*) ∨
  (*i*>*0* ∧ ((*ntaken i σ*) ⊨ *f*) ∧ (∀ *ia*<*i*. (*ntaken ia* (*ntaken i σ*) ⊨ ¬*f*)))
)
)
    by (*metis diff-zero enat-ord-simps*(*2*) *gr-zeroI less-numeral-extra*(*3*) *min.orderE ndropn-0*
       nlength-NNil nsubn-def1 nsubn-nlength-gr-one ntaken-0 ntaken-nlength)
hence *4*: (((*ntaken i σ*) ⊨ ▷*f*) ∧ *i*≤*nlength σ*) =
 (*i*≤*nlength σ* ∧ (
  (((*ntaken i σ*) ⊨ *f*) ∧ *i* = *0*) ∨
  (*i*>*0* ∧ ((*ntaken i σ*) ⊨ *f*) ∧ (∀ *ia*<*i*. (*ntaken ia σ* ⊨ ¬*f*)))
)
)
      using *less-imp-add-positive* **by** *fastforce*
 from *4* **show** *?thesis* **by** *auto*
qed

lemma *PrefixFstAndsem*:
 ((*ntaken i σ* ⊨ ▷*f* ∧ *g*) ∧ *i*≤*nlength σ*) =
  (*i*≤*nlength σ* ∧
   (
    ((*ntaken i σ* ⊨ *f* ∧ *g*) ∧ *i* = *0*) ∨
    (*i*>*0* ∧ (*ntaken i σ* ⊨ *f* ∧ *g*) ∧ (∀ *ia*<*i*. (*ntaken ia σ* ⊨ ¬*f*)))
)
)
using *PrefixFstsem*[*of f i σ*]   **by** (*metis unl-lift2*)

lemma *DiLenFstsem*:
 (*σ* ⊨ *di* (▷*f* ∧ *len*(*i*))) =
   (*i*≤*nlength σ* ∧
   (
    ((*ntaken i σ* ⊨ *f*) ∧ *i* = *0*) ∨
    (*i*>*0* ∧ (*ntaken i σ* ⊨ *f*) ∧ (∀ *ia*<*i*. (*ntaken ia σ* ⊨ ¬*f*)))
)

$$)$$
by (*simp add*: *DiLensem PrefixFstsem*)

lemma *DfLenFstsem*:
$$(\sigma \models df\ (\triangleright f \wedge len(i))) =$$
$$(\ i{\leq}nlength\ \sigma \wedge$$
$$($$
$$(\ (ntaken\ i\ \sigma \models f) \wedge i = 0) \vee$$
$$(\ i{>}0 \wedge (ntaken\ i\ \sigma \models f) \wedge (\forall\, ia{<}i.\ (ntaken\ ia\ \sigma \models \neg f)))$$
$$)$$
$$)$$
by (*simp add*: *DfLensem PrefixFstsem*)

lemma *DiLenFstAndsem*:
$$(\sigma \models di\ ((\triangleright f \wedge g) \wedge len(i))) =$$
$$(\ i{\leq}nlength\ \sigma \wedge$$
$$($$
$$(\ (ntaken\ i\ \sigma \models f \wedge g) \wedge i = 0) \vee$$
$$(\ i{>}0 \wedge (ntaken\ i\ \sigma \models f \wedge g) \wedge (\forall\, ia{<}i.\ (ntaken\ ia\ \sigma \models \neg f)))$$
$$)$$
$$)$$
using *DiLensem PrefixFstAndsem* **by** *metis*

lemma *DfLenFstAndsem*:
$$(\sigma \models df\ ((\triangleright f \wedge g) \wedge len(i))) =$$
$$(\ i{\leq}nlength\ \sigma \wedge$$
$$($$
$$(\ (ntaken\ i\ \sigma \models f \wedge g) \wedge i = 0) \vee$$
$$(\ i{>}0 \wedge (ntaken\ i\ \sigma \models f \wedge g) \wedge (\forall\, ia{<}i.\ (ntaken\ ia\ \sigma \models \neg f)))$$
$$)$$
$$)$$
using *DfLensem PrefixFstAndsem* **by** *metis*

lemma *FstLenSamesem*:
$$(\ (\ i{\leq}nlength\ \sigma \wedge$$
$$($$
$$(\ (ntaken\ i\ \sigma \models f) \wedge i = 0) \vee$$
$$(\ i{>}0 \wedge (ntaken\ i\ \sigma \models f) \wedge (\forall\, ia{<}i.\ (ntaken\ ia\ \sigma \models \neg f)))$$
$$)$$
$$) \wedge$$
$$(\ j{\leq}nlength\ \sigma \wedge$$
$$($$
$$(\ (ntaken\ j\ \sigma \models f) \wedge j = 0) \vee$$
$$(\ j{>}0 \wedge (ntaken\ j\ \sigma \models f) \wedge (\forall\, ia{<}j.\ (ntaken\ ia\ \sigma \models \neg f)))$$
$$)$$
$$) \longrightarrow (i{=}j)$$

by (*metis not-less-iff-gr-or-eq unl-lift*)

3.13.4 Theorems

Fixed length intervals

lemma *LenZeroEqvEmpty*:
$\vdash$ *len*(*0*) = *empty*
by (*simp add*: *len-d-def*)

lemma *LenOneEqvSkip*:
$\vdash$ *len*(*1*) = *skip*
by (*simp add*: *len-d-def ChopEmpty*)

lemma *LenNPlusOneA*:
$\vdash$ *len*(*n*+*1*) = *skip*;*len*(*n*)
by (*simp add*: *len-d-def*)

lemma *SkipFiniteEqvFiniteSkip*:
$\vdash$ *skip*;*finite* = *finite*;*skip*
using *FiniteChopSkipEqvSkipChopFinite* **by** *auto*

lemma *TrueEqvFiniteOrInfinite*:
$\vdash$ #*True* = *finite* $\vee$ *inf*
by (*simp add*: *FiniteOrInfinite*)

lemma *SkipInfEqvinfSkip*:
$\vdash$ *skip*;*inf* = *inf*;*skip*
by (*metis ChopAndInf MoreAndInfEqvInf MoreEqvSkipChopTrue PowerstarEqvSemhelp2 int-simps*(*16*) *inteq-reflection lift-and-com*)

lemma *SkipTrueEqvTrueSkip*:
$\vdash$ *skip*;#*True* = #*True*;*skip*
proof −
have *1*: $\vdash$ *skip*;#*True* = (*skip*;*finite* $\vee$ *skip*;*inf*)
by (*meson ChopOrEqv FiniteOrInfinite Prop04 RightChopEqvChop int-eq-true*)
have *2*: $\vdash$ (*finite*;*skip* $\vee$ *inf*;*skip*) = #*True*;*skip*
by (*meson FiniteOrInfinite LeftChopEqvChop OrChopEqv Prop04 int-eq-true*)
have *3*: $\vdash$ (*skip*;*finite* $\vee$ *skip*;*inf*) = (*finite*;*skip* $\vee$ *inf*;*skip*)
by (*metis 2 SkipFiniteEqvFiniteSkip SkipInfEqvinfSkip inteq-reflection*)
show *?thesis*
by (*metis 1 2 3 int-eq*)
qed

lemma *AndExistsLen*:
$\vdash$ (*f* $\wedge$ *finite*) = (*f* $\wedge$ ($\exists$ *k*. *len*(*k*)))
using *Finite-exist-len* **by** *fastforce*

lemma *AndExistsLenChop*:
$\vdash$ (*f*$\frown$*g*) = ($\exists$ *k*. (*f* $\wedge$ *len*(*k*));*g*)
by (*simp add*: *Valid-def len-defs chop-defs schop-defs*)

(*meson min.absorb1 nlength-eq-enat-nfiniteD*)

lemma *AndExistsLenSChop*:
$\vdash (f \frown g) = (\exists\, k.\ (f \wedge len(k)) \frown g)$
by (*simp add*: *Valid-def len-defs chop-defs schop-defs*)
(*meson min.absorb1 nlength-eq-enat-nfiniteD*)

lemma *AndExistsLenChopR*:
$\vdash (f;(g \wedge finite)) = (\exists\, k.\ f;(g \wedge len(k)))$
by (*simp add*: *Valid-def len-defs chop-defs finite-defs*)
(*metis ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn*)

lemma *AndExistsLenSChopR*:
$\vdash (f \frown (g \wedge finite)) = (\exists\, k.\ f \frown (g \wedge len(k)))$
by (*simp add*: *Valid-def len-defs schop-defs finite-defs*)
(*metis ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn*)

lemma *LFixedAndDistr*:
$\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g1) = ((f0 \wedge f1) \wedge len(k));(g0 \wedge g1)$
by (*auto simp add*: *Valid-def len-defs chop-defs nlength-eq-enat-nfiniteD*)

lemma *LFixedAndDistrS*:
$\vdash ((f0 \wedge len(k)) \frown g0 \wedge (f1 \wedge len(k)) \frown g1) = ((f0 \wedge f1) \wedge len(k)) \frown (g0 \wedge g1)$
by (*auto simp add*: *Valid-def len-defs schop-defs*)

lemma *RFixedAndDistr*:
$\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g1 \wedge len(k))) = (f0 \wedge f1);((g0 \wedge g1) \wedge len(k))$
unfolding *Valid-def itl-defs len-defs*
by *simp*
(*metis enat.simps(3) enat-add2-eq enat-add-sub-same le-iff-add ndropn-nlength*
nfinite-ndropn-b nlength-eq-enat-nfiniteD the-enat.simps)

lemma *RFixedAndDistrS*:
$\vdash (f0 \frown (g0 \wedge len(k)) \wedge f1 \frown (g1 \wedge len(k))) = (f0 \wedge f1) \frown ((g0 \wedge g1) \wedge len(k))$
by (*auto simp add*: *Valid-def len-defs schop-defs*)
(*metis enat.simps(3) enat-add2-eq enat-add-sub-same le-iff-add the-enat.simps*)

lemma *LFixedChopEqvFixedSChop*:
$\vdash (f \wedge len(k));g = (f \wedge len(k)) \frown g$
by (*auto simp add*: *Valid-def len-defs schop-defs chop-defs nlength-eq-enat-nfiniteD*)

lemma *LFixedAndDistrA*:
$\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g0) = ((f0 \wedge f1) \wedge len(k));g0$
proof −
have *1*: $\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g0) = ((f0 \wedge f1) \wedge len(k));(g0 \wedge g0)$
by (*rule LFixedAndDistr*)
have *2*: $\vdash ((f0 \wedge f1) \wedge len(k));(g0 \wedge g0) = ((f0 \wedge f1) \wedge len(k));g0$
by *auto*
from *1 2* **show** *?thesis* **by** *fastforce*

qed

lemma *LFixedAndDistrB*:
$\vdash ((f0 \wedge len(k));g0 \wedge (f0 \wedge len(k));g1) = (f0 \wedge len(k));(g0 \wedge g1)$
proof −
have *1*: $\vdash ((f0 \wedge len(k));g0 \wedge (f0 \wedge len(k));g1) = ((f0 \wedge f0) \wedge len(k));(g0 \wedge g1)$
by (*rule LFixedAndDistr*)
have *2*: $\vdash ((f0 \wedge f0) \wedge len(k));(g0 \wedge g1) = (f0 \wedge len(k));(g0 \wedge g1)$
by *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *LFixedAndDistrB1*:
$\vdash (len(k);f \wedge len(k);g) = len(k);(f \wedge g)$
proof −
have *1*: $\vdash len(k);f = (\#True \wedge len(k));f$
by *auto*
have *2*: $\vdash len(k);g = (\#True \wedge len(k));g$
by *auto*
have *3*: $\vdash (len(k);f \wedge len(k);g) = ((\#True \wedge len(k));f \wedge (\#True \wedge len(k));g)$
using *1 2* **by** *auto*
have *4*: $\vdash ((\#True \wedge len(k));f \wedge (\#True \wedge len(k));g) = (\#True \wedge len(k));(f \wedge g)$
using *LFixedAndDistrB* **by** *blast*
have *5*: $\vdash (\#True \wedge len(k));(f \wedge g) = (len(k));(f \wedge g)$
by *auto*
from *1 2 3 4 5* **show** *?thesis* **by** *auto*
qed

lemma *RFixedAndDistrA*:
$\vdash (f0;(g0 \wedge len(k)) \wedge f0;(g1 \wedge len(k))) = f0;((g0 \wedge g1) \wedge len(k))$
proof −
have *1*: $\vdash (f0;(g0 \wedge len(k)) \wedge f0;(g1 \wedge len(k))) = (f0 \wedge f0);((g0 \wedge g1) \wedge len(k))$
by (*rule RFixedAndDistr*)
have *2*: $\vdash (f0 \wedge f0);((g0 \wedge g1) \wedge len(k)) = f0;((g0 \wedge g1) \wedge len(k))$
by *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RFixedAndDistrAS*:
$\vdash (f0 \frown (g0 \wedge len(k)) \wedge f0 \frown (g1 \wedge len(k))) = f0 \frown ((g0 \wedge g1) \wedge len(k))$
proof −
have *1*: $\vdash (f0 \frown (g0 \wedge len(k)) \wedge f0 \frown (g1 \wedge len(k))) = (f0 \wedge f0) \frown ((g0 \wedge g1) \wedge len(k))$
by (*rule RFixedAndDistrS*)
have *2*: $\vdash (f0 \wedge f0) \frown ((g0 \wedge g1) \wedge len(k)) = f0 \frown ((g0 \wedge g1) \wedge len(k))$
by *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RFixedAndDistrB*:
$\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g0 \wedge len(k))) = (f0 \wedge f1);(g0 \wedge len(k))$

proof −
have *1*: ⊢ (*f0*;(*g0* ∧ *len*(*k*)) ∧ *f1*;(*g0* ∧ *len*(*k*))) = (*f0* ∧ *f1*);((*g0* ∧ *g0*) ∧ *len*(*k*))
by (*rule RFixedAndDistr*)
have *2*: ⊢ (*f0* ∧ *f1*);((*g0* ∧ *g0*) ∧ *len*(*k*)) = (*f0* ∧ *f1*);(*g0* ∧ *len*(*k*))
by *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RFixedAndDistrBS*:
⊢ (*f0*⌢(*g0* ∧ *len*(*k*)) ∧ *f1*⌢(*g0* ∧ *len*(*k*))) = (*f0* ∧ *f1*)⌢(*g0* ∧ *len*(*k*))
proof −
have *1*: ⊢ (*f0*⌢(*g0* ∧ *len*(*k*)) ∧ *f1*⌢(*g0* ∧ *len*(*k*))) = (*f0* ∧ *f1*)⌢((*g0* ∧ *g0*) ∧ *len*(*k*))
by (*rule RFixedAndDistrS*)
have *2*: ⊢ (*f0* ∧ *f1*)⌢((*g0* ∧ *g0*) ∧ *len*(*k*)) = (*f0* ∧ *f1*)⌢(*g0* ∧ *len*(*k*))
by *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopSkipAndChopSkip*:
⊢ (*f0*;*skip* ∧ *f1*;*skip*) = (*f0* ∧ *f1*);*skip*
proof −
have *1*: ⊢ (*f0*;(# *True* ∧ *len*(*1*)) ∧ *f1*;(# *True* ∧ *len*(*1*))) = (*f0* ∧ *f1*);(# *True* ∧ *len*(*1*))
by (*rule RFixedAndDistrB*)
have *2*: ⊢ (# *True* ∧ *len*(*1*)) = *skip*
using *LenOneEqvSkip* **by** *fastforce*
hence *3*: ⊢ *f0*;(# *True* ∧ *len*(*1*)) = *f0*;*skip*
using *RightChopEqvChop* **by** *blast*
have *4*: ⊢ *f1*;(# *True* ∧ *len*(*1*)) = *f1*;*skip*
using *2 RightChopEqvChop* **by** *blast*
have *5*: ⊢ (*f0*;(# *True* ∧ *len*(*1*)) ∧ *f1*;(# *True* ∧ *len*(*1*))) = (*f0*;*skip* ∧ *f1*;*skip*)
using *3 4* **by** *fastforce*
have *6*: ⊢ (*f0* ∧ *f1*);(# *True* ∧ *len*(*1*)) = (*f0* ∧ *f1*);*skip*
using *2 RightChopEqvChop* **by** *blast*
from *1 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopSkipAndSChopSkip*:
⊢ (*f0*⌢*skip* ∧ *f1*⌢*skip*) = (*f0* ∧ *f1*)⌢*skip*
proof −
have *1*: ⊢ (*f0*⌢(# *True* ∧ *len*(*1*)) ∧ *f1*⌢(# *True* ∧ *len*(*1*))) = (*f0* ∧ *f1*)⌢(# *True* ∧ *len*(*1*))
by (*rule RFixedAndDistrBS*)
have *2*: ⊢ (# *True* ∧ *len*(*1*)) = *skip*
using *LenOneEqvSkip* **by** *fastforce*
hence *3*: ⊢ *f0*⌢(# *True* ∧ *len*(*1*)) = *f0*⌢*skip*
using *RightSChopEqvSChop* **by** *blast*
have *4*: ⊢ *f1*⌢(# *True* ∧ *len*(*1*)) = *f1*⌢*skip*
using *2 RightSChopEqvSChop* **by** *blast*
have *5*: ⊢ (*f0*⌢(# *True* ∧ *len*(*1*)) ∧ *f1*⌢(# *True* ∧ *len*(*1*))) = (*f0*⌢*skip* ∧ *f1*⌢*skip*)
using *3 4* **by** *fastforce*
have *6*: ⊢ (*f0* ∧ *f1*)⌢(# *True* ∧ *len*(*1*)) = (*f0* ∧ *f1*)⌢*skip*

using *2 RightSChopEqvSChop* **by** *blast*
from *1 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BiAndChopSkipEqv*:
$\vdash (bi\ (f \wedge g));skip = ((bi\ f);skip \wedge (bi\ g);skip)$
proof −
have *1*: $\vdash bi\ (f \wedge g) = ((bi\ f) \wedge (bi\ g))$
by (*auto simp add*: *bi-defs Valid-def*)
hence *2*: $\vdash (bi\ (f \wedge g));skip = (bi\ f \wedge bi\ g);skip$
by (*rule LeftChopEqvChop*)
have *3*: $\vdash (bi\ f \wedge bi\ g);skip = ((bi\ f);skip \wedge (bi\ g);skip)$
using *ChopSkipAndChopSkip* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BfAndSChopSkipEqv*:
$\vdash (bf\ (f \wedge g))^\frown skip = ((bf\ f)^\frown skip \wedge (bf\ g)^\frown skip)$
proof −
have *1*: $\vdash bf\ (f \wedge g) = ((bf\ f) \wedge (bf\ g))$
by (*auto simp add*: *bf-defs Valid-def*)
hence *2*: $\vdash (bf\ (f \wedge g))^\frown skip = (bf\ f \wedge bf\ g)^\frown skip$
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (bf\ f \wedge bf\ g)^\frown skip = ((bf\ f)^\frown skip \wedge (bf\ g)^\frown skip)$
using *SChopSkipAndSChopSkip* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DiAndChopSkipEqv*:
$\vdash (di\ (f \wedge g));skip \longrightarrow (di\ f);skip \wedge (di\ g);skip$
proof −
have *1*: $\vdash di\ (f \wedge g) \longrightarrow (di\ f) \wedge (di\ g)$
by (*simp add*: *DiAndImpAnd*)
hence *2*: $\vdash (di\ (f \wedge g));skip \longrightarrow (di\ f \wedge di\ g);skip$
by (*rule LeftChopImpChop*)
have *3*: $\vdash (di\ f \wedge di\ g);skip = ((di\ f);skip \wedge (di\ g);skip)$
using *ChopSkipAndChopSkip* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfAndSChopSkipEqv*:
$\vdash (df\ (f \wedge g))^\frown skip \longrightarrow (df\ f)^\frown skip \wedge (df\ g)^\frown skip$
proof −
have *1*: $\vdash df\ (f \wedge g) \longrightarrow (df\ f) \wedge (df\ g)$
by (*simp add*: *DfAndImpAnd*)
hence *2*: $\vdash (df\ (f \wedge g))^\frown skip \longrightarrow (df\ f \wedge df\ g)^\frown skip$
by (*rule LeftSChopImpSChop*)
have *3*: $\vdash (df\ f \wedge df\ g)^\frown skip = ((df\ f)^\frown skip \wedge (df\ g)^\frown skip)$
using *SChopSkipAndSChopSkip* **by** *fastforce*

from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEmptyAndEmpty*:
$\vdash (f;g \wedge empty) = (f \wedge g \wedge empty)$
by (*simp add*: *Valid-def itl-defs*)
(*metis enat-0-iff*(*2*) *le-zero-eq ndropn-0 nfinite-ntaken ntaken-all*)

lemma *SChopEmptyAndEmpty*:
$\vdash (f \frown g \wedge empty) = (f \wedge g \wedge empty)$
by (*simp add*: *Valid-def itl-defs*)
(*metis is-NNil-ndropn le-numeral-extra*(*3*) *ndropn-0 ndropn-is-NNil ntaken-all zero-enat-def*)

lemma *ChopSkipImpMore*:
$\vdash f;skip \longrightarrow more$
by (*metis DiIntro DiSkipEqvMore RightChopImpMoreRule inteq-reflection*)

lemma *SChopSkipImpMore*:
$\vdash f \frown skip \longrightarrow more$
by (*metis DiIntro DiSkipEqvMore RightSChopImpMoreRule inteq-reflection*)

lemma *MoreEqvMoreChopTrue*:
$\vdash more = more;\#True$
proof −
have *1*: $\vdash more = skip;\#True$
using *MoreEqvSkipChopTrue* **by** *blast*
have *2*: $\vdash \#True = \#True;\#True$
by (*auto simp add*: *Valid-def chop-defs*)
(*metis zero-enat-def zero-le*)
hence *3*: $\vdash skip;\#True = skip;(\#True;\#True)$
using *RightChopEqvChop* **by** *blast*
have *4*: $\vdash skip;(\#True;\#True) = (skip;\#True);\#True$
using *ChopAssoc* **by** *blast*
have *5*: $\vdash (skip;\#True);\#True = more;\#True$
using *MoreEqvSkipChopTrue* **by** (*simp add*: *more-d-def next-d-def*)
from *1 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreEqvMoreSChopTrue*:
$\vdash more = more \frown \#True$
by (*metis MoreEqvSkipSChopTrue SChopAssoc TrueEqvTrueSChopTrue inteq-reflection*)

lemma *NotNotChopSkip*:
$\vdash (\neg((\neg f)\ ;skip)) = (empty \vee (f;skip))$
by (*metis WprevEqvEmptyOrPrev prev-d-def wprev-d-def*)

lemma *NotSChopFixed*:
$\vdash (\neg(f \frown (g \wedge len(k)))) = (\neg(\Diamond(g \wedge len(k))) \vee ((\neg f) \frown (g \wedge len(k))))$

by (*auto simp add*: *itl-defs Valid-def len-defs*)
(*metis diff-diff-cancel enat-ord-simps*(*1*) *idiff-enat-enat ndropn-nlength*
nfinite-conv-nlength-enat nfinite-ndropn the-enat.simps)

lemma *NotNotSChopSkip*:
$\vdash (\neg((\neg\ f) \frown skip)) = (empty \vee inf \vee (f \frown skip))$
proof −
have *1*: $\vdash (\ (\neg\ f) \frown skip \vee f \frown skip) = (\neg f \vee f) \frown skip$
by (*meson OrSChopEqv Prop11*)
have *2*: $\vdash (\neg f \vee f) = \#True$
by *simp*
have *3*: $\vdash more \wedge finite \longrightarrow \#True \frown skip$
by (*metis DiamondSChopdef FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip fmore-d-def*
int-simps(*13*) *int-simps*(*9*) *inteq-reflection sometimes-d-def*)
have *4*: $\vdash more \wedge finite \longrightarrow ((\neg\ f) \frown skip) \vee (f \frown skip)$
using *1 3* **by** *fastforce*
have *5*: $\vdash more \wedge finite \wedge \neg((f \frown skip)) \longrightarrow ((\neg\ f) \frown skip)$
using *4* **by** *fastforce*
have *6*: $\vdash (\neg((\neg\ f) \frown skip)) \longrightarrow empty \vee inf \vee (f \frown skip)$
using *5* **unfolding** *empty-d-def finite-d-def* **by** *fastforce*
have *7*: $\vdash ((\neg\ f) \frown skip) \longrightarrow finite$
by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)
have *8*: $\vdash ((\neg\ f) \frown skip) \longrightarrow more$
by (*simp add*: *SChopSkipImpMore*)
have *9*: $\vdash ((\neg\ f) \frown skip) \longrightarrow \neg((f \frown skip))$
by (*meson NotSChopSkipEqvFmoreAndNotSChopSkip Prop11 Prop12*)
have *10* :$\vdash ((\neg\ f) \frown skip) \longrightarrow more \wedge finite \wedge \neg((f \frown skip))$
by (*simp add*: *7 8 9 Prop12*)
have *11*: $\vdash (empty \vee inf \vee (f \frown skip)) \longrightarrow (\neg((\neg\ f) \frown skip))$
using *10* **unfolding** *finite-d-def empty-d-def* **by** *fastforce*
show *?thesis*
by (*meson 11 6 Prop11*)
qed

lemma *NotFixedSChop*:
$\vdash (\neg((g \wedge len(k)) \frown f)) = (\neg(df(g \wedge len(k))) \vee ((g \wedge len(k)) \frown (\neg f)))$
proof −
have *1*: $\vdash (g \wedge len(k)) \frown f = (g \wedge len(k)); f$
by (*meson LFixedChopEqvFixedSChop Prop11*)
have *2*: $\vdash (g \wedge len(k)) \frown (\neg f) = ((g \wedge len(k)); (\neg f))$
by (*meson LFixedChopEqvFixedSChop Prop11*)
have *3*: $\vdash df(g \wedge len(k)) = di(g \wedge len(k))$
by (*metis LFixedChopEqvFixedSChop df-d-def di-d-def inteq-reflection*)
show *?thesis* **using** *1 2 3*
by (*metis NotFixedChop inteq-reflection*)
qed

lemma *NotChopNotSkip*:
$\vdash (\neg(f; skip)) = (empty \vee ((\neg\ f); skip))$
proof −

have *1*: $\vdash (\neg((\neg(\neg f));skip)) = (empty \vee ((\neg f);skip))$ **using** *NotNotChopSkip* **by** *blast*
have *2*: $\vdash (\neg((\neg(\neg f));skip)) = (\neg(f;skip))$ **by** *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *NotSChopNotSkip*:
$\vdash (\neg(f \frown skip)) = (empty \vee inf \vee ((\neg f) \frown skip))$
by (*metis NotNotSChopSkip int-simps*(*4*) *inteq-reflection*)

lemma *NotNotSChopInf*:
$\vdash (\neg\ (\ (\neg f) \frown inf)) = (finite \vee (bf\ f))$
by (*auto simp add*: *Valid-def itl-defs*)

Additional ITL theorems

lemma *DiAndFiniteEqvDfAndFinite*:
$\vdash (di\ f \wedge finite) = (df\ f \wedge finite)$
by (*auto simp add*: *Valid-def di-defs df-defs finite-defs*)

lemma *DiEqvDfOrInf*:
$\vdash di\ f = (df\ f \vee (f \wedge inf))$
by (*simp add*: *ChopSChopdef df-d-def di-d-def*)

lemma *BiEqvBfAndfinite*:
$\vdash bi\ f = (bf\ f \wedge (inf \longrightarrow f))$
proof −
have *1*: $\vdash di\ (\neg f) = (df\ (\neg f) \vee \neg f \wedge inf)$
using *DiEqvDfOrInf*[*of LIFT* $\neg f$] **by** *blast*
have *2*: $\vdash (\neg di\ (\neg f)) = (\neg df\ (\neg f) \wedge (inf \longrightarrow f))$
using *1* **by** *fastforce*
show *?thesis*
unfolding *bf-d-def bi-d-def* **by** (*simp add*: *2*)
qed

lemma *DfEqvDiAndFinite*:
$\vdash df\ f = di\ (f \wedge finite)$
by (*simp add*: *df-d-def di-d-def schop-d-def*)

lemma *BfEqvBiOrInf*:
$\vdash bf\ f = bi\ (f \vee inf)$
proof −
have *1*: $\vdash df\ (\neg f) = di\ (\neg f \wedge finite)$
by (*simp add*: *DfEqvDiAndFinite*)
have *2*: $\vdash (\neg df\ (\neg f)) = (\neg di\ (\neg f \wedge finite))$
using *1* **by** *auto*
have *3*: $\vdash\ (\neg di\ (\neg f \wedge finite)) = (bi\ (\neg\ (\neg f \wedge finite)))$
by (*simp add*: *NotDiEqvBiNot*)
have *4*: $\vdash (\neg\ (\neg f \wedge finite)) = (f \vee inf)$

unfolding *finite-d-def* **by** *fastforce*
have *5*: $\vdash$ *bi* ($\neg$ ($\neg$ *f* $\wedge$ *finite*)) = *bi* (*f* $\vee$ *inf*)
by (*metis 3 4 inteq-reflection*)
have *6*: $\vdash$ *bf f* = ($\neg$ (*df* ($\neg$ *f*)))
by (*simp add*: *bf-d-def*)
show *?thesis*
by (*metis 2 3 5 6 inteq-reflection*)
qed

lemma *BiOrBiImpBiOr*:
$\vdash$ *bi f* $\vee$ *bi g* $\longrightarrow$ *bi*(*f* $\vee$ *g*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *f* $\vee$ *g* **by** *auto*
hence *2*: $\vdash$ *bi f* $\longrightarrow$ *bi*(*f* $\vee$ *g*) **by** (*rule BiImpBiRule*)
have *3*: $\vdash$ *g* $\longrightarrow$ *f* $\vee$ *g* **by** *auto*
hence *4*: $\vdash$ *bi g* $\longrightarrow$ *bi*(*f* $\vee$ *g*) **by** (*rule BiImpBiRule*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BfOrBfImpBfOr*:
$\vdash$ *bf f* $\vee$ *bf g* $\longrightarrow$ *bf*(*f* $\vee$ *g*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *f* $\vee$ *g* **by** *auto*
hence *2*: $\vdash$ *bf f* $\longrightarrow$ *bf*(*f* $\vee$ *g*) **by** (*rule BfImpBfRule*)
have *3*: $\vdash$ *g* $\longrightarrow$ *f* $\vee$ *g* **by** *auto*
hence *4*: $\vdash$ *bf g* $\longrightarrow$ *bf*(*f* $\vee$ *g*) **by** (*rule BfImpBfRule*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreAndBiImpBiChopSkip*:
$\vdash$ *more* $\wedge$ *bi f* $\longrightarrow$ (*bi f*);*skip*
proof −
have *1*: $\vdash$ (*bi f*);*skip* = (($\neg$(*di* ($\neg$ *f*)));*skip*) **by** (*simp add*: *bi-d-def*)
have *2*: $\vdash$ ($\neg$(($\neg$(*di* ($\neg$ *f*)));*skip*)) = (*empty* $\vee$ (*di* ($\neg$ *f*));*skip*) **by** (*rule NotNotChopSkip*)
have *3*: $\vdash$ *empty* $\longrightarrow$ *empty* $\vee$ *di* ($\neg$ *f*) **by** *auto*
have *4*: $\vdash$ (*di* ($\neg$ *f*));*skip* $\longrightarrow$ *di* ($\neg$ *f*) **using** *ChopImpDi DiEqvDiDi* **by** *fastforce*
hence *5*: $\vdash$ (*di* ($\neg$ *f*));*skip* $\longrightarrow$ *empty* $\vee$ *di* ($\neg$ *f*) **by** (*rule Prop05*)
have *6*: $\vdash$ $\neg$(($\neg$(*di* ($\neg$ *f*)));*skip*) $\longrightarrow$ *empty* $\vee$ *di* ($\neg$ *f*) **using** *2 3 5* **by** *fastforce*
hence *7*: $\vdash$ $\neg$(*empty* $\vee$ *di* ($\neg$ *f*)) $\longrightarrow$ $\neg$($\neg$(($\neg$(*di* ($\neg$ *f*)));*skip*)) **by** *fastforce*
have *8*: $\vdash$ ($\neg$($\neg$(($\neg$(*di* ($\neg$ *f*)));*skip*))) = (($\neg$(*di* ($\neg$ *f*)));*skip*) **by** *auto*
have *9*: $\vdash$ ($\neg$(*empty* $\vee$ *di* ($\neg$ *f*))) = (*more* $\wedge$ $\neg$(*di* ($\neg$ *f*)))
unfolding *empty-d-def* **by** *force*
have *10*: $\vdash$ (*more* $\wedge$ $\neg$(*di* ($\neg$ *f*))) = (*more* $\wedge$ *bi f*) **by** (*simp add*: *bi-d-def*)
from *1 6 7 8 9 10* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *MoreAndBiImpBiSChopSkip*:
$\vdash$ *more* $\wedge$ *finite* $\wedge$ *bi f* $\longrightarrow$ (*bi f*)$\frown$*skip*

proof −
have *1*: ⊢ (*bi f*)⌢*skip* = ((¬(*di* (¬ *f*)))⌢*skip*) **by** (*simp add*: *bi-d-def*)
have *2*: ⊢ (¬((¬(*di* (¬ *f*)))⌢*skip*)) = (*empty* ∨ *inf* ∨ (*di* (¬ *f*))⌢*skip*) **by** (*rule NotNotSChopSkip*)
have *3*: ⊢ *empty* ⟶ *empty* ∨ *di* (¬ *f*) **by** *auto*
have *03*: ⊢ *skip* ⟶ #*True* **by** *simp*
have *04*: ⊢ (*di* (¬ *f*))⌢*skip* ⟶ *di* (¬ *f*)⌢#*True* **by** (*simp add*: *RightSChopImpSChop*)
have *4*: ⊢ (*di* (¬ *f*))⌢*skip* ⟶ *df* (¬ *f*) **unfolding** *di-d-def df-d-def*
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc ChopImpDi ChopTrueAndFiniteEqvAndFiniteChopFinite di-d-def inteq-reflection schop-d-def*)
hence *5*: ⊢ (*di* (¬ *f*))⌢*skip* ⟶ *empty* ∨ *df* (¬ *f*) **by** (*rule Prop05*)
have *6*: ⊢ ¬((¬(*di* (¬ *f*)))⌢*skip*) ⟶ *empty* ∨ *inf* ∨ *df* (¬ *f*) **using** *2 3 5* **by** *fastforce*
hence *7*: ⊢ ¬(*empty* ∨ *inf* ∨ *df* (¬ *f*)) ⟶ ¬(¬((¬(*di* (¬ *f*)))⌢*skip*)) **by** *fastforce*
have *8*: ⊢ (¬(¬((¬(*di* (¬ *f*)))⌢*skip*))) = ((¬(*di* (¬ *f*)))⌢*skip*) **by** *auto*
have *9*: ⊢ (¬(*empty* ∨ *inf* ∨ *df* (¬ *f*))) = (*more* ∧ *finite* ∧¬(*df* (¬ *f*)))
unfolding *empty-d-def finite-d-def* **by** *force*
have *10*: ⊢ (*more* ∧ *finite* ∧ ¬(*df* (¬ *f*))) = (*more* ∧ *finite* ∧ *bf f*)
by (*simp add*: *bf-d-def*)
have *11*: ⊢ (*more* ∧ *finite* ∧ *bf f*) = (*more* ∧ *finite* ∧ *bi f*)
using *BfElim BiEqvBfAndfinite* **by** *fastforce*
from *1 6 7 8 9 10 11* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *MoreAndBfImpBfSChopSkip*:
⊢ *more* ∧ *bf f* ⟶ *inf* ∨ (*bf f*)⌢*skip*
proof −
have *1*: ⊢ (*bf f*)⌢*skip* = ((¬(*df* (¬ *f*)))⌢*skip*) **by** (*simp add*: *bf-d-def*)
have *2*: ⊢ (¬((¬(*df* (¬ *f*)))⌢*skip*)) = (*empty* ∨ *inf* ∨(*df* (¬ *f*))⌢*skip*) **by** (*rule NotNotSChopSkip*)
have *3*: ⊢ *empty* ⟶ *empty* ∨ *inf* ∨ *df* (¬ *f*) **by** *auto*
have *4*: ⊢ (*df* (¬ *f*))⌢*skip* ⟶ *df* (¬ *f*)
by (*metis DfEqvDfDf SChopImpDf inteq-reflection*)
hence *5*: ⊢ (*df* (¬ *f*))⌢*skip* ⟶ *empty* ∨ *inf* ∨ *df* (¬ *f*)
by (*simp add*: *Prop05*)
have *6*: ⊢ ¬((¬(*df* (¬ *f*)))⌢*skip*) ⟶ *empty* ∨ *inf* ∨ *df* (¬ *f*) **using** *2 3 5* **by** *fastforce*
hence *7*: ⊢ ¬(*empty* ∨ *inf* ∨ *df* (¬ *f*)) ⟶ ¬(¬((¬(*df* (¬ *f*)))⌢*skip*)) **by** *fastforce*
have *8*: ⊢ (¬(¬((¬(*df* (¬ *f*)))⌢*skip*))) = ((¬(*df* (¬ *f*)))⌢*skip*) **by** *auto*
have *9*: ⊢ (¬(*empty* ∨ *inf* ∨ *df* (¬ *f*))) = (*more* ∧ *finite* ∧¬(*df* (¬ *f*)))
unfolding *empty-d-def* **unfolding** *finite-d-def* **by** *fastforce*
have *10*: ⊢ (*more* ∧ *finite* ∧ ¬(*df* (¬ *f*))) = (*more* ∧ *finite* ∧ *bf f*) **by** (*simp add*: *bf-d-def*)
from *1 6 7 8 9 10* **show** *?thesis* **unfolding** *finite-d-def*
using *NotNotSChopSkip Prop11* **by** *fastforce*
qed

lemma *DiChopEqvChopDi*:
⊢ *di*(*f*;*g*) = *f*;(*di g*)
by (*metis ChopAssoc Prop11 di-d-def*)

lemma *DiChopImpDiB*:
⊢ *di*(*f*;*g*) ⟶ *di f*
proof −
have *1*: ⊢ *f* ; (*g*;#*True*) ⟶ *di f* **by** (*rule ChopImpDi*)

have *2*: $\vdash f\ ;\ (g;\#True) = (f;g);\#True$ **by** (*rule ChopAssoc*)
from *1 2* **show** *?thesis* **by** (*metis di-d-def int-eq*)
qed

lemma *DfSChopEqvSChopDf*:
$\vdash df(f;g) = f\frown(df\ g)$
by (*metis* (*no-types*, *opaque-lifting*) *ChopAndFiniteDist ChopEmpty SChopAssoc*
int-eq itl-def(*15*) *itl-def*(*9*))

lemma *DfChopEqvDfSchop*:
$\vdash df(f;g) = df(f\frown g)$
by (*metis DfSChopEqvSChopDf Prop04 Prop11 SChopAssoc df-d-def*)

lemma *DfSChopImpDfB*:
$\vdash df(f\frown g) \longrightarrow df\ f$
proof −
have *1*: $\vdash f \frown (g\frown\#True) \longrightarrow df\ f$ **by** (*simp add*: *SChopImpDf*)
have *2*: $\vdash f \frown (g\frown\#True) = (f\frown g)\frown\#True$ **by** (*rule SChopAssoc*)
from *1 2* **show** *?thesis* **by** (*metis df-d-def int-eq*)
qed

lemma *BiBiOrImpBi*:
$\vdash bi\ (\ bi\ f \vee bi\ g) \longrightarrow bi\ f \vee bi\ g$
using *BiElim* **by** *auto*

lemma *BfBfOrImpBf*:
$\vdash bf\ (\ bf\ f \vee bf\ g) \longrightarrow bf\ f \vee bf\ g$
proof −
have *1*: $\vdash bf\ (\ bf\ f \vee bf\ g) \wedge finite \longrightarrow bf\ f \vee bf\ g$
using *BfElim* **by** *fastforce*
have *2*: $\vdash bf\ (\ bf\ f \vee bf\ g) \wedge inf \longrightarrow bf\ f \vee bf\ g$
by (*auto simp add*: *Valid-def itl-defs*)
(*meson nle-le*)
show *?thesis* **using** *1 2* **unfolding** *finite-d-def*
by (*meson 1 OrFiniteInf Prop02 Prop11 lift-imp-trans*)
qed

lemma *BiImpBiBiOr*:
$\vdash bi\ f \longrightarrow bi\ (\ bi\ f \vee bi\ g)$
proof −
have *1*: $\vdash bi\ f \longrightarrow bi\ f \vee bi\ g$ **by** *auto*
hence *2*: $\vdash bi\ (bi\ f) \longrightarrow bi(bi\ f \vee bi\ g)$ **using** *BiImpBiRule* **by** *blast*
have *3*: $\vdash bi\ (bi\ f) = bi\ f$ **using** *BiEqvBiBi* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BfImpBfBfOr*:
$\vdash bf\ f \longrightarrow bf\ (\ bf\ f \vee bf\ g)$
proof −
have *1*: $\vdash bf\ f \longrightarrow bf\ f \vee bf\ g$ **by** *auto*

hence *2*: $\vdash$ *bf* (*bf f*) $\longrightarrow$ *bf*(*bf f* $\vee$ *bf g*) **using** *BfImpBfRule* **by** *blast*
have *3*: $\vdash$ *bf* (*bf f*) = *bf f* **using** *BfEqvBfBf* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiImpBiBiOrB*:
$\vdash$ *bi g* $\longrightarrow$ *bi* (*bi f* $\vee$ *bi g*)
proof −
have *1*: $\vdash$ *bi g* $\longrightarrow$ *bi f* $\vee$ *bi g* **by** *auto*
hence *2*: $\vdash$ *bi* (*bi g*) $\longrightarrow$ *bi*(*bi f* $\vee$ *bi g*) **using** *BiImpBiRule* **by** *blast*
have *3*: $\vdash$ *bi* (*bi g*) = *bi g* **using** *BiEqvBiBi* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BfImpBfBfOrB*:
$\vdash$ *bf g* $\longrightarrow$ *bf* (*bf f* $\vee$ *bf g*)
proof −
have *1*: $\vdash$ *bf g* $\longrightarrow$ *bf f* $\vee$ *bf g* **by** *auto*
hence *2*: $\vdash$ *bf* (*bf g*) $\longrightarrow$ *bf*(*bf f* $\vee$ *bf g*) **using** *BfImpBfRule* **by** *blast*
have *3*: $\vdash$ *bf* (*bf g*) = *bf g* **using** *BfEqvBfBf* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiBiOrEqvBi*:
$\vdash$ *bi* (*bi f* $\vee$ *bi g*) = (*bi f* $\vee$ *bi g*)
proof −
have *1*: $\vdash$ *bi* (*bi f* $\vee$ *bi g*) $\longrightarrow$ *bi f* $\vee$ *bi g* **by** (*rule BiBiOrImpBi*)
have *2*: $\vdash$ *bi f* $\longrightarrow$ *bi* (*bi f* $\vee$ *bi g*) **by** (*rule BiImpBiBiOr*)
have *3*: $\vdash$ *bi g* $\longrightarrow$ *bi* (*bi f* $\vee$ *bi g*) **by** (*rule BiImpBiBiOrB*)
have *4*: $\vdash$ *bi f* $\vee$ *bi g* $\longrightarrow$ *bi* (*bi f* $\vee$ *bi g*) **using** *2 3* **by** *fastforce*
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BfBfOrEqvBf*:
$\vdash$ *bf* (*bf f* $\vee$ *bf g*) = (*bf f* $\vee$ *bf g*)
by (*meson BfBfOrImpBf BfImpBfBfOr BfImpBfBfOrB Prop02 int-iffI*)

lemma *DiEqvOrDiChopSkipA*:
$\vdash$ *di f* = (*f* $\vee$ *di*(*f*;*skip*))
proof −
have *1*: $\vdash$ *di f* = *f* ;# *True* **by** (*simp add*: *di-d-def*)
hence *2*: $\vdash$ *di f* = *f*; (*empty* $\vee$ *more*) **by** (*simp add*: *empty-d-def*)
hence *3*: $\vdash$*f*; (*empty* $\vee$ *more*) = (*f*;*empty* $\vee$ *f*;*more*) **using** *ChopOrEqv* **by** *blast*
have *4*: $\vdash$ *f*;*empty* = *f* **by** (*rule ChopEmpty*)
have *5*: $\vdash$ *more* = *skip*;# *True* **using** *MoreEqvSkipChopTrue* **by** *blast*
hence *6*: $\vdash$ *f*;*more* = *f*;(*skip*;# *True*) **using** *RightChopEqvChop* **by** *blast*
have *7*: $\vdash$ *f*;(*skip*;# *True*) = (*f*;*skip*);# *True* **by** (*rule ChopAssoc*)
from *2 3 4 6 7* **show** *?thesis* **by** (*metis di-d-def int-eq*)
qed

lemma *DfEqvOrDfSChopSkipA*:
$\vdash df\ f = ((f \wedge finite) \vee df(f \frown skip))$
proof −
have *1*: $\vdash df\ f = f \frown \#True$ **by** (*simp add*: *df-d-def*)
hence *2*: $\vdash df\ f = f \frown (\ empty \vee more)$ **by** (*simp add*: *empty-d-def*)
hence *3*: $\vdash f \frown (\ empty \vee more) = (f \frown empty \vee f \frown more)$ **using** *SChopOrEqv* **by** *blast*
have *4*: $\vdash f \frown empty = (f \wedge finite)$ **by** (*simp add*: *ChopEmpty schop-d-def*)
have *5*: $\vdash more = skip \frown \#True$ **using** *MoreEqvSkipSChopTrue* **by** *simp*
hence *6*: $\vdash f \frown more = f \frown (skip \frown \#True)$ **using** *RightSChopEqvSChop* **by** *blast*
have *7*: $\vdash f \frown (skip \frown \#True) = (f \frown skip) \frown \#True$ **by** (*rule SChopAssoc*)
from *2 3 4 6 7* **show** *?thesis* **by** (*metis df-d-def inteq-reflection*)
qed

lemma *DiEqvOrDiChopSkipB*:
$\vdash di\ f = (f \vee (di\ f); skip)$
proof −
have *1*: $\vdash (di\ f) = (f \vee di(f; skip))$ **by** (*rule DiEqvOrDiChopSkipA*)
have *2*: $\vdash\ di(f; skip) = (f; skip); \#True$ **by** (*simp add*: *di-d-def*)
have *3*: $\vdash (f; skip); \#True = f; (skip; \#True)$
by (*meson ChopAssoc Prop11*)
have *4*: $\vdash di(f; skip) = f; (skip; \#True)$ **using** *2 3* **by** *fastforce*
have *5*: $\vdash skip; \#True = \#True; skip$
by (*simp add*: *SkipTrueEqvTrueSkip*)
hence *6*: $\vdash f; (skip; \#True) = f; (\#True; skip)$ **using** *RightChopEqvChop* **by** *blast*
have *7*: $\vdash di(f; skip) = f; (\#True; skip)$ **using** *4 6* **by** *fastforce*
have *8*: $\vdash f; (\#True; skip) = (f; \#True); skip$ **by** (*rule ChopAssoc*)
have *9*: $\vdash (f; \#True); skip = (di\ f); skip$ **by** (*simp add*: *di-d-def*)
have *10* : $\vdash di(f; skip) = (di\ f); skip$ **using** *7 8 9* **by** *fastforce*
hence *11*: $\vdash (f \vee di(f; skip)) = (f \vee (di\ f); skip)$ **by** *auto*
from *1 11* **show** *?thesis* **by** *fastforce*
qed

lemma *DfEqvOrDfSChopSkipB*:
$\vdash df\ f = ((f \wedge finite) \vee (df\ f) \frown skip \vee (df\ f) \frown inf)$
proof −
have *1*: $\vdash (df\ f) = ((f \wedge finite) \vee df(f \frown skip))$ **by** (*rule DfEqvOrDfSChopSkipA*)
have *2*: $\vdash\ df(f \frown skip) = (f \frown skip) \frown \#True$ **by** (*simp add*: *df-d-def*)
have *3*: $\vdash (f \frown skip) \frown \#True = f \frown (skip \frown \#True)$
by (*meson SChopAssoc Prop11*)
have *4*: $\vdash df(f \frown skip) = f \frown (skip \frown \#True)$ **using** *2 3* **by** *fastforce*
have *5*: $\vdash skip \frown \#True = (\#True \frown skip \vee \#True \frown inf)$
by (*metis FiniteChopInfEqvInf FmoreEqvSkipChopFinite MoreAndInfEqvInf MoreEqvSkipSChopTrue*
OrFiniteInf SkipFiniteEqvFiniteSkip fmore-d-def int-simps(*17*) *inteq-reflection schop-d-def*)
hence *6*: $\vdash f \frown (skip \frown \#True) = f \frown (\#True \frown skip \vee \#True \frown inf)$ **using** *RightSChopEqvSChop* **by** *blast*
have *7*: $\vdash df(f \frown skip) = f \frown (\#True \frown skip \vee \#True \frown inf)$ **using** *4 6* **by** *fastforce*
have *71*: $\vdash f \frown (\#True \frown skip \vee \#True \frown inf) = (\ f \frown (\#True \frown skip) \vee f \frown (\#True \frown inf))$
by (*simp add*: *SChopOrEqv*)
have *8*: $\vdash f \frown (\#True \frown skip) = (f \frown \#True) \frown skip$ **by** (*rule SChopAssoc*)
have *81*: $\vdash f \frown (\#True \frown inf) = (f \frown \#True) \frown inf$ **by** (*rule SChopAssoc*)

have *9*: $\vdash (f \frown \# True) \frown skip = (df\ f) \frown skip$ **by** (*simp add*: *df-d-def*)
have *91*: $\vdash (f \frown \# True) \frown inf = (df\ f) \frown inf$ **by** (*simp add*: *df-d-def*)
have *10* : $\vdash df(f \frown skip) = ((df\ f) \frown skip \vee (df\ f) \frown inf)$
using *8 81 9 91 7 71* **by** (*metis int-eq*)
hence *11*: $\vdash ((f \wedge finite) \vee df(f \frown skip)) = ((f \wedge finite) \vee (df\ f) \frown skip \vee (df\ f) \frown inf)$ **by** *auto*
from *1 11* **show** *?thesis* **by** *fastforce*
qed

lemma *BiEqvAndEmptyOrBiChopSkip*:
$\vdash bi\ f = (f \wedge (empty \vee (bi\ f); skip))$
proof −
have *1*: $\vdash di\ (\neg f) = (\neg f \vee (di\ (\neg f); skip))$ **by** (*rule DiEqvOrDiChopSkipB*)
have *2*: $\vdash di\ (\neg f) = (\neg(\ bi\ f\ ))$ **by** (*rule DiNotEqvNotBi*)
have *3*: $\vdash (\neg(\ bi\ f\ )) = (\neg f \vee (di\ (\neg f); skip))$ **using** *1 2* **by** *fastforce*
hence *4*: $\vdash bi\ f = (\neg(\neg f \vee (di\ (\neg f); skip)))$ **by** *auto*
have *5*: $\vdash (\neg(\neg f \vee (di\ (\neg f); skip))) = (f \wedge \neg(di\ (\neg f); skip))$ **by** *auto*
have *6*: $\vdash di\ (\neg f); skip = ((\neg(bi\ f)); skip)$ **by** (*simp add*: *2 LeftChopEqvChop*)
hence *7*: $\vdash (\neg(di\ (\neg f); skip)) = (\neg((\neg(bi\ f)); skip))$ **by** *auto*
have *8*: $\vdash (\neg((\neg(bi\ f)); skip)) = (empty \vee (bi\ f); skip)$ **using** *NotNotChopSkip* **by** *blast*
hence *9*: $\vdash (f \wedge \neg(di\ (\neg f); skip)) = (f \wedge (empty \vee (bi\ f); skip))$ **using** *7 8* **by** *fastforce*
from *4 5 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BfEqvAndEmptyOrBfSChopSkip*:
$\vdash bf\ f = (\ (finite \longrightarrow f) \wedge (empty \vee inf \vee (bf\ f) \frown skip) \wedge (finite \vee bf\ f)\ )$
proof −
have *1*: $\vdash (df\ (\neg f)) = ((\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf))$
by (*rule DfEqvOrDfSChopSkipB*)
have *2*: $\vdash (df\ (\neg f)) = (\neg(\ bf\ f\ ))$ **by** (*rule DfNotEqvNotBf*)
have *3*: $\vdash (\neg(\ bf\ f\ )) = (\ (\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf))$
using *1 2* **by** *fastforce*
hence *4*: $\vdash (bf\ f) = (\neg((\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf)))$ **by** *auto*
have *5*: $\vdash (\neg((\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf))) =$
$(\ (finite \longrightarrow f) \wedge (\neg(df\ (\neg f) \frown skip)) \wedge (\neg(df\ (\neg f) \frown inf)))$ **by** *auto*
have *50*: $\vdash (bf\ f) = (\ (finite \longrightarrow f) \wedge \neg(df\ (\neg f) \frown skip) \wedge \neg(df\ (\neg f) \frown inf))$
using *4* **by** *auto*
have *6*: $\vdash df\ (\neg f) \frown skip = ((\neg(bf\ f)) \frown skip)$ **by** (*simp add*: *2 LeftSChopEqvSChop*)
hence *7*: $\vdash (\neg(df\ (\neg f) \frown skip)) = (\neg((\neg(bf\ f)) \frown skip))$ **by** *auto*
have *8*: $\vdash (\neg((\neg(bf\ f)) \frown skip)) = (empty \vee inf \vee (bf\ f) \frown skip)$ **using** *NotNotSChopSkip* **by** *blast*
have *61*: $\vdash df\ (\neg f) \frown inf = ((\neg(bf\ f)) \frown inf)$ **by** (*simp add*: *2 LeftSChopEqvSChop*)
hence *71*: $\vdash (\neg(df\ (\neg f) \frown inf)) = (\neg((\neg(bf\ f)) \frown inf))$ **by** *auto*
have *81*: $\vdash (\neg(df\ (\neg f) \frown inf)) = (finite \vee (bf\ (bf\ f)))$
by (*metis 2 NotNotSChopInf int-eq*)
have *82*: $\vdash (bf\ (bf\ f)) = bf\ f$
by (*simp add*: *BfEqvBfBf BfImpBfBf int-iffD2 int-iffI*)
have *83*: $\vdash (finite \vee (bf\ (bf\ f))) = (finite \vee bf\ f)$
using *82* **by** *auto*

hence *9*: $\vdash ((\mathit{finite} \longrightarrow f) \wedge \neg(\mathit{df}\ (\neg\ f) \frown \mathit{skip}) \wedge\ \neg(\mathit{df}\ (\neg\ f) \frown \mathit{inf})) =$
$((\mathit{finite}\ \longrightarrow f) \wedge (\mathit{empty} \vee \mathit{inf} \vee (\mathit{bf}\ f) \frown \mathit{skip}) \wedge (\mathit{finite} \vee \mathit{bf}\ f)\ )$
by (*metis 2 8 81 inteq-reflection lift-and-com*)
from *4 5 9* **show** *?thesis*
by (*metis inteq-reflection*)
qed

lemma *DiDiAndEqvDi*:
$\vdash \mathit{di}\ (\ \mathit{di}\ f \wedge \mathit{di}\ g) = (\mathit{di}\ f \wedge \mathit{di}\ g)$
proof −
have *1*: $\vdash \mathit{bi}\ (\ \mathit{bi}\ (\neg\ f) \vee \mathit{bi}\ (\neg\ g)) = (\mathit{bi}\ (\neg f) \vee \mathit{bi}\ (\neg\ g))$
by (*meson BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB Prop02 int-iffI*)
have *2*: $\vdash \mathit{bi}\ (\neg f) = (\neg\ (\mathit{di}\ f))$
by (*simp add*: *bi-d-def*)
have *3*: $\vdash \mathit{bi}\ (\neg g) = (\neg\ (\mathit{di}\ g))$
by (*simp add*: *bi-d-def*)
have *4*: $\vdash (\mathit{bi}\ (\neg f) \vee \mathit{bi}\ (\neg\ g)) = (\neg\ (\mathit{di}\ f) \vee \neg\ (\mathit{di}\ g))$
using *2 3* **by** *fastforce*
have *5*: $\vdash (\neg\ (\mathit{di}\ f) \vee \neg\ (\mathit{di}\ g)) = (\neg(\mathit{di}\ f \wedge \mathit{di}\ g))$
by *auto*
have *6*: $\vdash \mathit{bi}\ (\ \mathit{bi}\ (\neg\ f) \vee \mathit{bi}\ (\neg\ g)) = (\neg(\mathit{di}\ f \wedge \mathit{di}\ g))$
using *1 5 4* **by** *fastforce*
hence *7*: $\vdash (\neg(\mathit{bi}\ (\ \mathit{bi}\ (\neg\ f) \vee \mathit{bi}\ (\neg\ g)))) = (\mathit{di}\ f \wedge \mathit{di}\ g)$
by *auto*
have *8* : $\vdash (\neg(\mathit{bi}\ (\ \mathit{bi}\ (\neg\ f) \vee \mathit{bi}\ (\neg\ g)))) = \mathit{di}\ (\ \neg(\mathit{bi}\ (\neg\ f) \vee \mathit{bi}\ (\neg\ g)))$
using *DiNotEqvNotBi* **by** *fastforce*
have *9* : $\vdash (\neg(\mathit{bi}\ (\neg\ f) \vee \mathit{bi}\ (\neg\ g))) = (\mathit{di}\ f \wedge \mathit{di}\ g)$
using *1 7* **by** *fastforce*
hence *10*: $\vdash \mathit{di}\ (\ \neg(\mathit{bi}\ (\neg\ f) \vee \mathit{bi}\ (\neg\ g))) = \mathit{di}\ (\ \mathit{di}\ f \wedge \mathit{di}\ g)$
using *DiEqvDi* **by** *blast*
from *7 8 10* **show** *?thesis* **by** *fastforce*
qed

lemma *DfDfAndEqvDf*:
$\vdash \mathit{df}\ (\ \mathit{df}\ f \wedge \mathit{df}\ g) = (\mathit{df}\ f \wedge \mathit{df}\ g)$
proof −
have *1*: $\vdash \mathit{bf}\ (\ \mathit{bf}\ (\neg\ f) \vee \mathit{bf}\ (\neg\ g)) = (\mathit{bf}\ (\neg f) \vee \mathit{bf}\ (\neg\ g))$
by (*meson BfBfOrImpBf BfImpBfBfOr BfImpBfBfOrB Prop02 int-iffI*)
have *2*: $\vdash \mathit{bf}\ (\neg f) = (\neg\ (\mathit{df}\ f))$
by (*simp add*: *bf-d-def*)
have *3*: $\vdash \mathit{bf}\ (\neg g) = (\neg\ (\mathit{df}\ g))$
by (*simp add*: *bf-d-def*)
have *4*: $\vdash (\mathit{bf}\ (\neg f) \vee \mathit{bf}\ (\neg\ g)) = (\neg\ (\mathit{df}\ f) \vee \neg\ (\mathit{df}\ g))$
using *2 3* **by** *fastforce*
have *5*: $\vdash (\neg\ (\mathit{df}\ f) \vee \neg\ (\mathit{df}\ g)) = (\neg(\mathit{df}\ f \wedge \mathit{df}\ g))$
by *auto*
have *6*: $\vdash \mathit{bf}\ (\ \mathit{bf}\ (\neg\ f) \vee \mathit{bf}\ (\neg\ g)) = (\neg(\mathit{df}\ f \wedge \mathit{df}\ g))$
using *1 5 4* **by** *fastforce*
hence *7*: $\vdash (\neg(\mathit{bf}\ (\ \mathit{bf}\ (\neg\ f) \vee \mathit{bf}\ (\neg\ g)))) = (\mathit{df}\ f \wedge \mathit{df}\ g)$

by *auto*
have *8* : $\vdash (\neg(bf\ (\ bf\ (\neg\ f) \vee bf\ (\neg\ g)))) = df\ (\ \neg(bf\ (\neg\ f) \vee bf\ (\neg\ g)))$
using *DfNotEqvNotBf* **by** *fastforce*
have *9* : $\vdash (\neg(bf\ (\neg\ f) \vee bf\ (\neg\ g))) = (df\ f \wedge df\ g)$
using *1 7* **by** *fastforce*
hence *10*: $\vdash df\ (\ \neg(bf\ (\neg\ f) \vee bf\ (\neg\ g))) = df\ (\ df\ f \wedge df\ g)$
using *DfEqvDf* **by** *blast*
from *7 8 10* **show** *?thesis* **by** *fastforce*
qed

lemma *BiInductLen*:
$\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge (len\ k) \longrightarrow bi\ f$
proof −
have *1*: $\vdash len\ k \longrightarrow finite$
by (*simp add*: *len-k-finite*)
have *2*: $\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge finite \longrightarrow bi\ f$
by (*simp add*: *BiInduct*)
have *3*: $\vdash finite = (\exists\, k.\ len\ k)$
by (*simp add*: *Finite-exist-len*)
have *4*: $\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge (\exists\, k.\ len\ k) \longrightarrow bi\ f$
by (*metis 2 3 inteq-reflection*)
show *?thesis* **using** *4* **by** *fastforce*
qed

lemma *NotFinitePrevEqv*:
$\vdash (\neg((f \wedge finite);skip)) = (empty \vee (\neg\ f \vee inf);skip)$
proof −
have *1*: $\vdash (\neg((f \wedge finite);skip)) = (empty \vee (\neg(f \wedge finite));skip)$
by (*simp add*: *NotChopNotSkip*)
have *2*: $\vdash (\neg(f \wedge finite)) = (\neg f \vee inf)$
unfolding *finite-d-def* **by** *fastforce*
have *3*: $\vdash (empty \vee (\neg(f \wedge finite));skip) = (empty \vee (\neg f \vee inf);skip)$
by (*metis 1 2 inteq-reflection*)
show *?thesis*
by (*metis 1 3 inteq-reflection*)
qed

lemma *WPrevAndFiniteEqv*:
$\vdash wprev\ (f \wedge finite) = (empty \vee f \frown skip)$
unfolding *wprev-d-def prev-d-def schop-d-def*
by (*simp add*: *NotNotChopSkip*)

lemma *BiInductB*:
$\vdash bi(f \longrightarrow wprev\ (f \wedge finite)) \wedge f \wedge finite \longrightarrow bi\ f$
unfolding *Valid-def itl-defs*
by *simp*
(*metis biinduct-help nfinite-conv-nlength-enat*)

lemma *BfInduct*:
$\vdash bf(f \longrightarrow wprev\ f) \wedge f\ \wedge finite \longrightarrow bf\ f$
by (*auto simp add*: *Valid-def itl-defs*)
(*metis biinduct-help nfinite-conv-nlength-enat*)

lemma *PrevLoop*:
assumes $\vdash f \longrightarrow prev\ f$
shows $\vdash finite \longrightarrow \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow prev\ f$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow (\ more \wedge wprev\ f)$
by (*metis ChopSkipImpMore Prop05 Prop12 WprevEqvEmptyOrPrev inteq-reflection*
lift-imp-trans prev-d-def)
hence *3*: $\vdash f \longrightarrow wprev\ f$ **by** *auto*
hence *4*: $\vdash bi(f \longrightarrow wprev\ f)$ **by** (*rule BiGen*)
have *5*: $\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge finite \longrightarrow bi\ f$ **by** (*rule BiInduct*)
hence *6*: $\vdash bi(f \longrightarrow wprev\ f) \longrightarrow (f \wedge finite \longrightarrow bi\ f)$ **by** *fastforce*
have *7*: $\vdash (f \wedge finite \longrightarrow bi\ f)$ **using** *4 6 MP* **by** *blast*
have *8*: $\vdash bi\ f \longrightarrow f$ **by** (*rule BiElim*)
have *9*: $\vdash finite \longrightarrow f\ = bi\ f$ **using** *7 8* **by** *fastforce*
have *10*: $\vdash f \longrightarrow more$ **using** *2* **by** *auto*
hence *11*: $\vdash bi\ f \longrightarrow\ bi\ more$ **using** *BiImpBiRule* **by** *blast*
have *12*: $\vdash \neg(bi\ more)$ **using** *DiEmpty bi-d-def empty-d-def* **by** (*simp add*: *bi-d-def empty-d-def*)
from *7 9 11 12* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *FinitePrevLoop*:
assumes $\vdash f \longrightarrow prev\ (f \wedge finite)$
shows $\vdash finite \longrightarrow \neg\ f$
by (*metis AndChopA PrevLoop assms lift-imp-trans prev-d-def*)

lemma *PrevImpNotPrevNot*:
$\vdash prev\ f \longrightarrow \neg(prev\ (\neg\ f))$
by (*metis NotNotChopSkip Prop03 prev-d-def*)

lemma *BiEqvAndWprevBi*:
$\vdash bi\ f = (f \wedge wprev(bi\ f))$
by (*metis BiEqvAndEmptyOrBiChopSkip WprevEqvEmptyOrPrev inteq-reflection prev-d-def*)

lemma *DiIntroB*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow prev\ f$
shows $\vdash f \wedge finite \longrightarrow di\ g$
proof −
have *1*: $\vdash f \wedge \neg\ \ g \longrightarrow prev\ f$
using *assms* **by** *auto*

hence *2*: $\vdash f \wedge \neg\ g \wedge (bi\ (\neg\ g)) \longrightarrow (prev\ f) \wedge (bi\ (\neg\ g))$
by *auto*
have *3*: $\vdash (bi\ (\neg\ g)) \longrightarrow \neg\ g$
by (*rule BiElim*)
hence *4*: $\vdash bi\ (\neg\ g) = ((bi\ (\neg\ g)) \wedge \neg\ g)$
by *fastforce*
have *5*: $\vdash f \wedge (bi\ (\neg\ g)) \longrightarrow prev\ f \wedge bi\ (\neg\ g)$
using *2 4* **by** *fastforce*
have *6*: $\vdash bi\ (\neg\ g) = ((\neg\ g) \wedge wprev(bi\ (\neg\ g)))$
using *BiEqvAndWprevBi* **by** *blast*
have *7*: $\vdash prev\ f \wedge bi\ (\neg\ g) \longrightarrow\ prev\ f \wedge wprev(bi\ (\neg\ g))$
using *6* **by** *auto*
have *8*: $\vdash f \wedge (bi\ (\neg\ g)) \longrightarrow prev\ f \wedge wprev(bi\ (\neg\ g))$
using *5 7* **using** *lift-imp-trans* **by** *blast*
hence *9*: $\vdash f \wedge (bi\ (\neg\ g)) \longrightarrow more \wedge wprev\ f \wedge wprev(bi\ (\neg\ g))$
using *zero-enat-def* **by** (*auto simp*: *Valid-def itl-defs*)
(*metis le-zero-eq nlength-eq-enat-nfiniteD*)
hence *10*: $\vdash f \wedge (bi\ (\neg\ g)) \longrightarrow wprev\ f \wedge wprev(bi\ (\neg\ g))$
by *auto*
hence *11*: $\vdash f \wedge (bi\ (\neg\ g)) \longrightarrow wprev\ (f \wedge bi\ (\neg\ g))$
by (*auto simp*: *Valid-def itl-defs*)
hence *12*: $\vdash bi(f \wedge (bi\ (\neg\ g)) \longrightarrow wprev\ (f \wedge bi\ (\neg\ g)))$
by (*rule BiGen*)
have *13*: $\vdash bi(f \wedge (bi\ (\neg\ g)) \longrightarrow wprev\ (f \wedge bi\ (\neg\ g))) \wedge$
$(f\ \wedge (bi\ (\neg\ g))) \wedge finite \longrightarrow bi(f \wedge (bi\ (\neg\ g)))$
using *BiInduct* **by** *auto*
hence *14*: $\vdash bi(f \wedge (bi\ (\neg\ g)) \longrightarrow wprev\ (f \wedge bi(\neg\ g))) \longrightarrow$
$((f \wedge (bi(\neg\ g))) \wedge finite \longrightarrow bi(f \wedge (bi\ (\neg\ g))))$
by *fastforce*
have *15*: $\vdash ((f \wedge (bi\ (\neg\ g))) \wedge finite \longrightarrow bi(f \wedge (bi\ (\neg\ g))))$
using *12 14 MP* **by** *blast*
have *16*: $\vdash bi(f \wedge (bi\ (\neg\ g))) \longrightarrow (f \wedge (bi\ (\neg\ g)))$
by (*rule BiElim*)
have *17*: $\vdash finite \longrightarrow bi(f \wedge (bi\ (\neg\ g))) = (f \wedge (bi\ (\neg\ g)))$
using *16 15* **by** *fastforce*
have *18*: $\vdash (f \wedge (bi\ (\neg\ g))) \longrightarrow more$
using *9* **by** *auto*
hence *19*: $\vdash bi(f \wedge (bi\ (\neg\ g))) \longrightarrow bi\ more$
by (*simp add*: *BiImpBiRule*)
have *191*: $\vdash (\neg\ more) = empty$
by (*simp add*: *empty-d-def*)
have *20*: $\vdash (\neg(bi\ more))$
unfolding *bi-d-def di-d-def*
by (*metis EmptyChop TrueW empty-d-def int-simps*(*2*) *int-simps*(*3*) *inteq-reflection*)
have *21*: $\vdash\ finite \longrightarrow \neg(f \wedge (bi\ (\neg\ g)))$
using *17 19 20* **by** *fastforce*
hence *22*: $\vdash\ finite \longrightarrow \neg\ f \vee \neg\ (bi\ (\neg\ g))$
by *auto*
have *23*: $\vdash (\neg\ (bi\ (\neg\ g))) = di\ g$
by (*auto simp*: *bi-d-def*)

show *?thesis* **using** *22 23* **by** *fastforce*
qed

lemma *DiIntroC*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow prev\ (f \wedge finite)$
shows $\vdash f \wedge finite \longrightarrow di\ g$
using *assms*
by (*metis AndChopA DiIntroB lift-imp-trans prev-d-def*)

lemma *DfIntroB*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow prev\ f$
shows $\vdash f \wedge finite \longrightarrow df\ g$
proof −
have *1*: $\vdash f \wedge finite \longrightarrow di\ g$
by (*simp add*: *DiIntroB assms*)
have *2*: $\vdash di\ g = (df\ g \vee (g \wedge inf))$
by (*simp add*: *DiEqvDfOrInf*)
have *3*: $\vdash f \wedge finite \longrightarrow finite \wedge (df\ g \vee (g \wedge inf))$
by (*metis 1 ChopSChopdef FiniteImp df-d-def di-d-def int-eq lift-and-com*)
have *4*: $\vdash finite \wedge (df\ g \vee (g \wedge inf)) \longrightarrow df\ g$
unfolding *finite-d-def* **by** *fastforce*
show *?thesis*
using *3 4 lift-imp-trans* **by** *blast*
qed

lemma *DfIntroC*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow prev\ (f \wedge finite)$
shows $\vdash f \wedge finite \longrightarrow df\ g$
using *assms*
by (*metis AndChopA DfIntroB lift-imp-trans prev-d-def*)

lemma *DiEqvOrChopMore*:
$\vdash di\ f = (f \vee f;more)$
proof −
have *1*: $\vdash di\ f = f;\#True$ **by** (*simp add*: *di-d-def*)
hence *2*: $\vdash di\ f = f;\ (empty \vee more)$ **by** (*simp add*: *empty-d-def*)
have *3*: $\vdash f;\ (empty \vee more) = (f;empty \vee f;more)$ **by** (*simp add*: *ChopOrEqv*)
have *4*: $\vdash f;empty = f$ **by** (*rule ChopEmpty*)
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DfEqvOrSChopMore*:
$\vdash df\ f = ((f \wedge finite) \vee f \frown more)$
proof −
have *1*: $\vdash df\ f = f \frown \#True$ **by** (*simp add*: *df-d-def*)
hence *2*: $\vdash df\ f = f \frown\ (empty \vee more)$ **by** (*simp add*: *empty-d-def*)
have *3*: $\vdash f \frown\ (empty \vee more) = (f \frown empty \vee f \frown more)$ **by** (*simp add*: *SChopOrEqv*)
have *4*: $\vdash f \frown empty = (f \wedge finite)$ **by** (*simp add*: *ChopEmpty schop-d-def*)
from *2 3 4* **show** *?thesis* **by** *fastforce*

qed

lemma *DiAndDiEqvDiAndDiOrDiAndDi*:
$\vdash (di\ f \wedge di\ g) = (di(f \wedge di\ g) \vee di(g \wedge di\ f))$
unfolding *Valid-def*
apply *auto*
unfolding *itl-defs*
apply *simp*
apply (*metis min.absorb1 nle-le*)
apply *simp*
apply (*metis*)
apply *simp*
apply (*metis min.absorb1*)
apply *simp*
apply (*metis min.absorb1*)
apply *simp*
apply (*metis*)
done

lemma *DfAndDfEqvDiAndDfOrDfAndDf*:
$\vdash (df\ f \wedge df\ g) = (df(f \wedge df\ g) \vee df(g \wedge df\ f))$
unfolding *Valid-def itl-defs*
using *linorder-le-cases* **by** *fastforce*

lemma *BoxStateEqvBiFinState*:
$\vdash \Box\ (init\ w) = bi\ (finite \longrightarrow fin\ (init\ w))$
proof −
have *1*: $\vdash \Diamond\ (\neg\ (init\ w)) = finite\ ;\ (\neg(init\ w))$
by (*simp add*: *sometimes-d-def*)
have *2*: $\vdash \Diamond\ (init(\neg\ w)) = finite\ ;\ init\ (\neg\ w)$
by (*simp add*: *sometimes-d-def*)
have *3*: $\vdash di\ (finite \wedge fin\ (init\ (\neg\ w))) = finite\ ;\ init\ (\neg\ w)$
by (*metis DiAndFinEqvChopState NowImpDiamond Prop10 inteq-reflection lift-and-com*)
have *4*: $\vdash \Diamond\ (init(\neg\ w)) = di\ (finite \wedge fin\ (init\ (\neg\ w)))$
using *1 2 3* **by** *fastforce*
have *5*: $\vdash (\neg\ (\Diamond\ (init(\neg\ w)))) = (\neg\ (di\ (finite \wedge fin\ (init\ (\neg\ w)))))$
using *4* **by** *fastforce*
have *51*: $\vdash (\neg(finite \wedge fin\ (init\ (\neg\ w)))) = (finite \longrightarrow \neg\ (fin\ (init\ (\neg\ w))))$
by *fastforce*
have *6*: $\vdash \Box\ (init\ w) = (\neg\ (di\ (finite \wedge fin\ (init\ (\neg\ w)))))$
using *5 always-d-def Initprop*(*2*) **by** (*metis int-eq*)
have *7*: $\vdash \Box\ (init\ w) = bi\ (finite \longrightarrow \neg\ (fin\ (init\ (\neg\ w))))$
using *6 51* **unfolding** *bi-d-def*
by (*metis int-simps*(*4*) *inteq-reflection*)
have *8*: $\vdash init\ (\neg\ w) = (\neg\ (init\ w))$
using *Initprop*(*2*) **by** *fastforce*
have *9*: $\vdash fin\ (init\ (\neg\ w)) = fin\ (\neg\ (init\ w))$
using *8 FinEqvFin* **by** *blast*

```
  have 10: ⊢ finite ⟶fin (init (¬ w)) = (¬ (fin (init w)))
      using 8 FinNotStateEqvNotFinState FinEqvFin  by fastforce
  have 11: ⊢ finite ⟶ ((¬ (fin (init (¬ w)))) = (fin (init w)))
      using 10  by fastforce
  have 111: ⊢ (finite ⟶ ¬ (fin (init (¬ w)))) = (finite ⟶fin (init w))
    using 11 by fastforce
  have 12: ⊢ bi (finite ⟶ ¬ (fin (init (¬ w)))) = bi (finite ⟶fin (init w))
      using 111  by (simp add: BiEqvBi)
  have 13: ⊢ □ (init w) = bi (finite ⟶fin (init w))
      using 7 12 by fastforce
  from 13 show ?thesis by simp
qed

lemma BoxStateEqvBfFinState:
⊢ □ (init w) = bf (fin (init w))
proof −
 have 1: ⊢ bi (finite ⟶fin (init w)) =
          (bf (finite ⟶fin (init w)) ∧ (inf ⟶ finite ⟶fin (init w)))
  by (simp add: BiEqvBfAndfinite)
 have 2: ⊢ (bf (finite ⟶fin (init w)) ∧ (inf ⟶ finite ⟶fin (init w))) =
            bf (finite ⟶fin (init w))
   unfolding finite-d-def by fastforce
 have 3: ⊢ □ (init w) = bi (finite ⟶ fin (init w))
   by (simp add: BoxStateEqvBiFinState)
 have 4: ⊢ bf (finite ⟶fin (init w)) = bf (fin (init w))
    by (simp add: FiniteImpBfEqvRule intI)
 show ?thesis
 by (metis 1 2 3 4 inteq-reflection)
qed


lemma DiamondStateEqvDiFinState:
⊢ ◇ (init w) = di (finite ∧ fin (init w))
proof −
  have 1: ⊢ □ (init (¬ w)) = bi (finite⟶fin (init (¬ w)))
      using BoxStateEqvBiFinState by blast
  have 2: ⊢ (¬ (□ (init (¬ w)))) = (¬ (bi (finite ⟶fin (init (¬ w)))))
      using 1 by auto
  have 3: ⊢ ◇ (¬ (init (¬ w))) = di (finite ∧ ¬ (fin (init (¬ w))))
      using 2
      by (metis (no-types, opaque-lifting) FinChopEqvDiamond FinNotStateEqvNotFinState
       Initprop(2) di-d-def int-simps(17) inteq-reflection lift-and-com)
  have 4: ⊢ ◇ (init  w) = di (finite ∧ ¬ (fin (init (¬ w))))
      by (metis 3 DiEqvNotBiNot DiState Initprop(2) StateEqvBi int-eq)
  have 5: ⊢ ◇ (init  w) = di (finite ∧ fin (init w)) using 4 FinNotStateEqvNotFinState
    by (metis int-simps(4) inteq-reflection lift-and-com)
  from 1 2 3 4 5 show ?thesis by simp
qed

lemma DiamondStateEqvDfFinState:
```

$\vdash \Diamond\ (init\ w) = df\ (fin\ (init\ w))$
by (*metis DiamondStateEqvDiFinState df-d-def di-d-def inteq-reflection*
lift-and-com schop-d-def)

lemma *OrDiEqvDi*:
$\vdash (f \vee di\ f) = di\ f$
by (*simp add*: *Prop05 Prop11 DiIntro*)

lemma *OrDfEqvDf*:
$\vdash ((f \wedge finite) \vee df\ f) = df\ f$
by (*simp add*: *AndFiniteImpDf Prop05 Prop11*)

lemma *AndDiEqv*:
$\vdash (f \wedge di\ f) = f$
proof −
have *1*: $\vdash f \longrightarrow di\ f$ **using** *DiIntro* **by** *blast*
from *1* **show** *?thesis* **by** *auto*
qed

lemma *AndDfEqv*:
$\vdash ((f \wedge finite) \wedge df\ f) = (f \wedge finite)$
by (*meson AndFiniteImpDf Prop10 Prop11*)

lemma *BiEmptyEqvEmpty*:
$\vdash bi\ empty = empty$
proof −
have *1*: $\vdash bi\ empty = (\neg\ (di\ (\neg\ empty)))$ **by** (*simp add*: *bi-d-def*)
have *2*: $\vdash (\neg\ (di\ (\neg\ empty))) = (\neg\ ((\neg\ empty);\#True))$ **by** (*simp add*: *di-d-def*)
have *3*: $\vdash (\neg\ ((\neg\ empty);\#True)) = (\neg\ (more;\#True))$ **by** (*simp add*: *empty-d-def*)
have *4*: $\vdash more;\#True = more$ **using** *MoreEqvMoreChopTrue* **by** *auto*
hence *5*: $\vdash (\neg(more;\#True)) = (\neg\ more)$ **by** *fastforce*
from *1 2 3 5* **show** *?thesis* **by** (*metis empty-d-def inteq-reflection*)
qed

lemma *BfEmptyEqvEmpty*:
$\vdash bf\ empty = empty$
by (*metis MoreEqvMoreSChopTrue empty-d-def int-eq int-simps*(*4*) *itl-def*(*15*) *itl-def*(*23*))

lemma *EmptyChopSkipInduct*:
assumes $\vdash empty \longrightarrow f$
$\vdash prev\ f \longrightarrow f$
shows $\vdash finite \longrightarrow f$
proof −
have *1*: $\vdash empty \longrightarrow f$ **using** *assms*(*1*) **by** *auto*
have *2*: $\vdash prev\ f \longrightarrow f$ **using** *assms*(*2*) **by** *blast*
have *3*: $\vdash (empty \vee prev\ f) \longrightarrow f$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash wprev\ f = (empty \vee prev\ f)$ **by** (*simp add*: *WprevEqvEmptyOrPrev*)
hence *5*: $\vdash wprev\ f \longrightarrow f$ **using** *3* **by** *fastforce*

hence *6*: $\vdash \neg f \longrightarrow \neg\ (wprev\ f)$ **by** *fastforce*
hence *7*: $\vdash \neg f \longrightarrow prev\ (\neg\ f)$ **by** (*simp add*: *wprev-d-def*)
hence *8*: $\vdash finite \longrightarrow \neg\ \neg\ f$ **by** (*rule PrevLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *PrevOrInfEqv*:
$\vdash prev\ (f \vee inf) = (prev\ (f \wedge finite) \vee inf)$
proof −
have *1*: $\vdash prev\ (f \vee inf)\ = (f \vee inf);skip$
unfolding *prev-d-def* **by** *auto*
have *2*: $\vdash (f \vee inf);skip = (f;skip \vee inf;skip)$
by (*simp add*: *OrChopEqv*)
have *3*: $\vdash f;skip = ((f \wedge finite);skip \vee (f \wedge inf);skip)$
by (*simp add*: *OrChopEqvRule OrFiniteInf*)
have *4*: $\vdash (f \wedge inf);skip = (f \wedge inf)$
by (*simp add*: *AndInfChopEqvAndInf*)
have *5*: $\vdash inf;skip = inf$
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf int-eq*)
have *6*: $\vdash ((f \wedge inf) \vee inf) = (\ inf)$
by *fastforce*
have *7*: $\vdash (f;skip \vee inf;skip) = ((f \wedge finite);skip \vee (f \wedge inf) \vee inf)$
using *3 4 5 6* **by** *fastforce*
have *8* : $\vdash ((f \wedge finite);skip \vee (f \wedge inf) \vee inf) =$
$((f \wedge finite);skip \vee\ \ inf)$
using *6* **by** *fastforce*
show *?thesis*
by (*metis 2 7 8 inteq-reflection prev-d-def*)
qed

lemma *EmptySChopSkipInduct*:
assumes $\vdash empty \longrightarrow f$
$\vdash prev\ (f \vee\ \ inf) \longrightarrow f$
shows $\vdash finite \longrightarrow f$
proof −
have *1*: $\vdash empty\ \longrightarrow f$ **using** *assms(1)* **by** *auto*
have *2*: $\vdash prev\ (f \vee inf) \longrightarrow f$ **using** *assms(2)* **by** *blast*
have *3*: $\vdash (empty \vee prev\ (f \vee inf)) \longrightarrow f$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash wprev\ (f \vee inf) = (empty \vee prev\ (f \vee inf))$ **by** (*simp add*: *WprevEqvEmptyOrPrev*)
hence *5*: $\vdash wprev\ (f \vee inf) \longrightarrow f$ **using** *3* **by** *fastforce*
hence *6*: $\vdash \neg f \longrightarrow \neg\ (wprev\ (f \vee inf))$ **by** *fastforce*
hence *7*: $\vdash \neg f \longrightarrow prev\ (\neg\ (f \vee inf))$ **by** (*simp add*: *wprev-d-def*)
have *8* : $\vdash (\neg\ (f \vee inf)) = (\neg\ f \wedge finite)$ **unfolding** *finite-d-def* **by** *fastforce*
hence *9*: $\vdash finite \longrightarrow \neg\ \neg\ f$ **using** *FinitePrevLoop*[*of LIFT* $\neg\ f$]
by (*metis 7 inteq-reflection*)
from *9* **show** *?thesis* **by** *auto*
qed

lemma *EmptySChopSkipInductB*:

assumes $\vdash$ *empty* $\longrightarrow$ *f*
$\vdash$ *prev* (*f* $\wedge$ *finite*) $\longrightarrow$ *f*
shows $\vdash$ *finite* $\longrightarrow$ *f*
by (*metis EmptyChopSkipInduct EmptyImpFinite Prop12 SChopImpFinite WPowerstar-ext WPowerstar-skip-finite assms(1) assms(2) int-eq itl-def(14) itl-def(9)*)

lemma *MoreImpImpChopSkipEqv*:
$\vdash$ *more* $\longrightarrow$ ((*f*$\longrightarrow$*g*);*skip* = ((*f*;*skip*)$\longrightarrow$(*g*;*skip*)))
proof −
have *01*: $\vdash$ (*f*$\longrightarrow$*g*) = ($\neg$ *f* $\vee$ *g*) **by** *auto*
hence *02*: $\vdash$ (*f*$\longrightarrow$*g*);*skip* = ($\neg$ *f* $\vee$ *g*);*skip* **by** (*simp add*: *LeftChopEqvChop*)
hence *1*: $\vdash$ (*more* $\wedge$ (*f*$\longrightarrow$*g*);*skip*) = (*more* $\wedge$ ($\neg$ *f* $\vee$ *g*);*skip*) **by** *fastforce*
have *2*: $\vdash$ ($\neg$ *f* $\vee$ *g*);*skip* = (($\neg$ *f*);*skip* $\vee$ *g*;*skip*)
using *OrChopEqv* **by** *auto*
hence *3*: $\vdash$ (*more* $\wedge$ ($\neg$ *f* $\vee$ *g*);*skip*) = (*more* $\wedge$ (($\neg$ *f*);*skip* $\vee$ *g*;*skip*))
by *auto*
have *4*: $\vdash$ ($\neg$(($\neg$ *f*);*skip*)) = (*empty* $\vee$ (*f*;*skip*))
using *NotNotChopSkip* **by** *blast*
hence *5*: $\vdash$ (($\neg$ *f*);*skip*) = ($\neg$(*empty* $\vee$ (*f*;*skip*)))
by *fastforce*
have *6*: $\vdash$ $\neg$(*empty* $\vee$ (*f*;*skip*)) = (*more* $\wedge$ $\neg$(*f*;*skip*))
using *5 NotChopSkipEqvMoreAndNotChopSkip* **by** *fastforce*
have *7*: $\vdash$ (($\neg$ *f*);*skip* $\vee$ *g*;*skip*) = ((*more* $\wedge$ $\neg$(*f*;*skip*)) $\vee$ *g*;*skip*)
using *5 6* **by** *fastforce*
hence *8*: $\vdash$ (*more* $\wedge$($\neg$ *f*;*skip* $\vee$ *g*;*skip*)) = (*more* $\wedge$ ((*more* $\wedge$ $\neg$(*f*;*skip*)) $\vee$ *g*;*skip*))
by *auto*
have *9*: $\vdash$ (*more* $\wedge$ ((*more* $\wedge$ $\neg$(*f*;*skip*)) $\vee$ *g*;*skip*)) = (*more* $\wedge$ ($\neg$(*f*;*skip*) $\vee$ *g*;*skip*))
by *auto*
have *10*: $\vdash$ (*more* $\wedge$ ($\neg$(*f*;*skip*) $\vee$ *g*;*skip*)) = (*more* $\wedge$ ((*f*;*skip*)$\longrightarrow$(*g*;*skip*)))
by *auto*
have *11*: $\vdash$ (*more* $\wedge$ (*f*$\longrightarrow$*g*);*skip*) = (*more* $\wedge$ ((*f*;*skip*)$\longrightarrow$(*g*;*skip*)))
using *1 2 3 8 9 10 7* **by** *fastforce*
from *11* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *MoreImpImpSChopSkipEqv*:
$\vdash$ *more* $\wedge$ *finite* $\longrightarrow$ ((*f*$\longrightarrow$*g*)$\frown$*skip* = ((*f*$\frown$*skip*)$\longrightarrow$(*g*$\frown$*skip*)))
proof −
have *01*: $\vdash$ (*f*$\longrightarrow$*g*) = ($\neg$ *f* $\vee$ *g*) **by** *auto*
hence *02*: $\vdash$ (*f*$\longrightarrow$*g*)$\frown$*skip* = ($\neg$ *f* $\vee$ *g*)$\frown$*skip* **by** (*simp add*: *LeftSChopEqvSChop*)
have *03*: $\vdash$ (*f*$\longrightarrow$*g*)$\frown$*skip* $\longrightarrow$ *finite*
by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)
have *04*: $\vdash$ (*more* $\wedge$ (*f*$\longrightarrow$*g*)$\frown$*skip*) = (*more* $\wedge$ *finite* $\wedge$ (*f*$\longrightarrow$*g*)$\frown$*skip*)
using *03* **by** *auto*
have *1*: $\vdash$ (*more* $\wedge$ *finite* $\wedge$ (*f*$\longrightarrow$*g*)$\frown$*skip*) = (*more* $\wedge$ *finite* $\wedge$ ($\neg$ *f* $\vee$ *g*)$\frown$*skip*)
by (*metis 01 04 inteq-reflection*)
have *2*: $\vdash$ ($\neg$ *f* $\vee$ *g*)$\frown$*skip* = (($\neg$ *f*)$\frown$*skip* $\vee$ *g*$\frown$*skip*)
using *OrSChopEqv* **by** *auto*

hence *3*: $\vdash$ (*more* $\wedge$ *finite* $\wedge$ ($\neg$ *f* $\vee$ *g*)$\frown$*skip*) = (*more* $\wedge$ *finite* $\wedge$ (($\neg$ *f*)$\frown$*skip* $\vee$ *g*$\frown$*skip*))
by *auto*
have *4*: $\vdash$ ($\neg$(($\neg$ *f*)$\frown$*skip*)) = (*empty* $\vee$ *inf* $\vee$ (*f*$\frown$*skip*))
using *NotNotSChopSkip* **by** *blast*
hence *5*: $\vdash$ (($\neg$ *f*)$\frown$*skip*) = ($\neg$(*empty* $\vee$ *inf* $\vee$ (*f*$\frown$*skip*)))
by *fastforce*
have *6*: $\vdash$ $\neg$(*empty* $\vee$ *inf* $\vee$(*f*$\frown$*skip*)) = (*more* $\wedge$ *finite* $\wedge$ $\neg$(*f*$\frown$*skip*))
using *5* **unfolding** *finite-d-def empty-d-def* **by** *fastforce*
have *7*: $\vdash$ (($\neg$ *f*)$\frown$*skip* $\vee$ *g*$\frown$*skip*) = ((*more* $\wedge$ *finite* $\wedge$ $\neg$(*f*$\frown$*skip*)) $\vee$ *g*$\frown$*skip*)
using *5 6* **by** *fastforce*
hence *8*: $\vdash$ (*more* $\wedge$ *finite* $\wedge$ ($\neg$ (*f*$\frown$*skip*) $\vee$ *g*$\frown$*skip*)) =
(*more* $\wedge$ *finite* $\wedge$ ((*more* $\wedge$ *finite* $\wedge$ $\neg$(*f*$\frown$*skip*)) $\vee$ *g*$\frown$*skip*))
by *fastforce*
have *9*: $\vdash$ (*more* $\wedge$ *finite* $\wedge$ ((*more* $\wedge$ *finite* $\wedge$ $\neg$(*f*$\frown$*skip*)) $\vee$ *g*$\frown$*skip*)) =
(*more* $\wedge$ *finite* $\wedge$ ($\neg$(*f*$\frown$*skip*) $\vee$ *g*$\frown$*skip*))
by *auto*
have *10*: $\vdash$ (*more* $\wedge$ *finite* $\wedge$ ($\neg$(*f*$\frown$*skip*) $\vee$ *g*$\frown$*skip*)) =
(*more* $\wedge$ *finite* $\wedge$ ((*f*$\frown$*skip*)$\longrightarrow$(*g*$\frown$*skip*)))
by *auto*
have *11*: $\vdash$ (*more* $\wedge$ *finite* $\wedge$ (*f*$\longrightarrow$*g*)$\frown$*skip*) = (*more* $\wedge$ *finite* $\wedge$ ((*f*$\frown$*skip*)$\longrightarrow$(*g*$\frown$*skip*)))
using *1 2 3 8 9 10 7*
by (*metis inteq-reflection*)
from *11* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *MoreImpImpPrevEqv*:
$\vdash$ *more* $\longrightarrow$ (*prev*(*f*$\longrightarrow$*g*) = (*prev f*$\longrightarrow$ *prev g*))
by (*simp add*: *MoreImpImpChopSkipEqv prev-d-def*)

lemma *BiBoxNotEqvNotFiniteChopChopTrue*:
$\vdash$ *bi*($\Box$ ($\neg$ *f*)) = ($\neg$((*finite*;*f*);# *True*))
by (*simp add*: *bi-d-def always-d-def di-d-def sometimes-d-def*)

lemma *BfBoxNotEqvNotTrueSChopSChopTrue*:
$\vdash$ *bf*($\Box$ ($\neg$ *f*)) = ($\neg$((# *True*$\frown$*f*)$\frown$# *True*))
by (*simp add*: *bf-d-def always-d-def df-d-def sometimes-d-def schop-d-def*)

lemma *DiAndEmptyEqvAndEmpty*:
$\vdash$ (*di f* $\wedge$ *empty*) = (*f* $\wedge$ *empty*)
by (*metis ChopEmptyAndEmpty di-d-def int-simps*(*17*) *inteq-reflection*)

lemma *DfAndEmptyEqvAndEmpty*:
$\vdash$ (*df f* $\wedge$ *empty*) = (*f* $\wedge$ *empty*)
by (*metis ChopEmptyAndEmpty DiAndEmptyEqvAndEmpty Prop04 SChopEmptyAndEmpty df-d-def di-d-def*)

lemma *FiniteImpBfEqvBi*:
$\vdash$ *finite* $\longrightarrow$ *bf f* = *bi f*
proof −
have *1*: $\vdash$ *bi f* = (*bf f* $\wedge$ (*inf* $\longrightarrow$ *f*))
by (*simp add*: *BiEqvBfAndfinite*)
have *2*: $\vdash$ *finite* $\longrightarrow$ (*bf f* $\wedge$ (*inf* $\longrightarrow$ *f*)) = *bf f*
unfolding *finite-d-def* **by** *fastforce*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *FiniteImpDfEqvDi*:
$\vdash$ *finite* $\longrightarrow$ *df f* = *di f*
proof −
have *1*: $\vdash$ *di f* = (*df f* $\vee$ (*f* $\wedge$ *inf*))
by (*simp add*: *DiEqvDfOrInf*)
have *2*: $\vdash$ *finite* $\longrightarrow$ (*df f* $\vee$ (*f* $\wedge$ *inf*)) = *df f*
unfolding *finite-d-def* **by** *fastforce*
show *?thesis* **using** *1 2* **by** *fastforce*
qed

Strict initial intervals

lemma *BsEqvEmptyOrBiSChopSkip*:
$\vdash$ *bs f* = (*empty* $\vee$ ((*bi f*)⌢*skip*))
unfolding *bs-d-def* **by** *simp*

lemma *DsMoreDi*:
$\vdash$ *ds f* = (*more* $\wedge$ (*finite* $\longrightarrow$ (*di f*)⌢*skip*))
proof −
have *1*: $\vdash$ *ds f* = (¬(*bs* (¬ *f*)))
by (*simp add*: *ds-d-def*)
have *2*: $\vdash$ (¬(*bs* (¬ *f*))) = (¬(*empty* $\vee$ (*bi* (¬ *f*))⌢*skip*))
by (*simp add*: *bs-d-def*)
have *3*: $\vdash$ (¬(*empty* $\vee$ (*bi* (¬ *f*))⌢*skip*)) = (¬*empty* $\wedge$ ¬((*bi* (¬*f*))⌢*skip*))
by *auto*
have *4*: $\vdash$ (¬*empty* $\wedge$ ¬((*bi* (¬*f*))⌢*skip*)) = (*more* $\wedge$ ¬((*bi* (¬*f*))⌢*skip*))
unfolding *empty-d-def* **by** *auto*
have *40*: $\vdash$ (*bi* (¬*f*)) = (¬(*di f*))
by (*meson NotDiEqvBiNot Prop11*)
have *41*: $\vdash$ (¬((*bi* (¬*f*))⌢*skip*)) = (*empty* $\vee$ *inf* $\vee$ ((*di f*)⌢*skip*))
using *40 NotNotSChopSkip*[*of LIFT* (*di f*)] **by** (*metis inteq-reflection*)
have *5*: $\vdash$ (*more* $\wedge$ ¬((*bi* (¬*f*))⌢*skip*)) = (*more* $\wedge$ (*empty* $\vee$ *inf* $\vee$ (*di f*)⌢*skip*))
using *41* **by** *auto*
have *6*: $\vdash$ (*more* $\wedge$ (*empty* $\vee$ *inf* $\vee$ (*di f*)⌢*skip*)) =
(*more* $\wedge$ (*inf* $\vee$ (*di f*)⌢*skip*))
unfolding *empty-d-def* **by** *fastforce*
show *?thesis* **unfolding** *finite-d-def* **using** *1 2 3 4 5*
by *fastforce*
qed

lemma *DsDi*:
$\vdash$ *ds f* = (*finite* $\longrightarrow$ (*di f*)$\frown$*skip*)
proof −
have *1*: $\vdash$ *ds f* = (*more* $\wedge$ (*finite* $\longrightarrow$ (*di f*)$\frown$*skip*)) **by** (*rule DsMoreDi*)
have *2*: $\vdash$ (*di f*)$\frown$*skip* $\longrightarrow$ *more* **by** (*simp add*: *SChopSkipImpMore*)
hence *3*: $\vdash$ (*more* $\wedge$ (*finite* $\longrightarrow$ (*di f*)$\frown$*skip*)) =
((*more* $\wedge$ *inf*) $\vee$ (*more* $\wedge$ (*di f*)$\frown$*skip*))
unfolding *finite-d-def* **by** *fastforce*
have *4*: $\vdash$ ((*more* $\wedge$ *inf*) $\vee$ (*more* $\wedge$ (*di f*)$\frown$*skip*)) =
(*inf* $\vee$ (*di f*)$\frown$*skip*)
by (*metis 2 3 MoreAndInfEqvInf Prop10 inteq-reflection lift-and-com*)
from *1 2 4* **show** *?thesis* **unfolding** *finite-d-def*
by *fastforce*
qed

lemma *DsDf*:
$\vdash$ *ds f* = (*finite* $\longrightarrow$ (*df f*)$\frown$*skip*)
proof −
have *1*: $\vdash$ *finite* $\longrightarrow$ (*di f*) = (*df f*)
using *FiniteImpDfEqvDi* **by** *fastforce*
have *2*: $\vdash$ *ds f* = (*finite* $\longrightarrow$ (*di f*)$\frown$*skip*)
by (*simp add*: *DsDi*)
have *3*: $\vdash$ (*finite* $\longrightarrow$ (*di f*)$\frown$*skip*) = (*finite* $\longrightarrow$ (*df f*)$\frown$*skip*)
by (*metis 1 2 FiniteImpAnd inteq-reflection schop-d-def*)
show *?thesis*
by (*metis 2 3 inteq-reflection*)
qed

lemma *BsEqvNotDsNot*:
$\vdash$ *bs f* = (¬(*ds* (¬ *f*)))
proof −
have *1*: $\vdash$ *ds* (¬ *f*) = (*more* $\wedge$ (*finite* $\longrightarrow$ *di* (¬ *f*)$\frown$*skip*))
using *DsMoreDi*[*of LIFT* ¬ *f*] **by** *blast*
hence *2*: $\vdash$ (¬(*ds* (¬ *f*))) = (¬(*more* $\wedge$ (*finite* $\longrightarrow$ *di* (¬ *f*)$\frown$*skip*)))
by *auto*
have *3*: $\vdash$ (¬(*more* $\wedge$ (*finite* $\longrightarrow$ *di* (¬ *f*)$\frown$*skip*))) =
(*empty* $\vee$ ¬(*finite* $\longrightarrow$ (*di* (¬ *f*))$\frown$*skip*))
unfolding *empty-d-def* **by** *fastforce*
have *31*: $\vdash$ *di* (¬ *f*) = (¬(*bi f*))
by (*simp add*: *DiNotEqvNotBi*)
have *4*: $\vdash$ (*empty* $\vee$ ¬(*finite* $\longrightarrow$ (*di* (¬ *f*))$\frown$*skip*)) =
(*empty* $\vee$ (*finite* $\wedge$ ¬((*di* (¬ *f*))$\frown$*skip*)))
by *fastforce*
have *41*: $\vdash$ (¬((*di* (¬ *f*))$\frown$*skip*)) = (¬((¬(*bi f*))$\frown$ *skip*))
by (*metis 31 int-simps*(*15*) *inteq-reflection*)
have *5*: $\vdash$ (¬((¬(*bi f*))$\frown$*skip*)) = (*empty* $\vee$ *inf* $\vee$ (*bi f*)$\frown$*skip*)
by (*rule NotNotSChopSkip*)
have *51*: $\vdash$ (*empty* $\vee$ (*finite* $\wedge$ ¬((*di* (¬ *f*))$\frown$*skip*))) =
(*empty* $\vee$ (*finite* $\wedge$ (*empty* $\vee$ *inf* $\vee$ (*bi f*)$\frown$*skip*)))

by (*metis 31 4 5 inteq-reflection*)
have *52*: ⊢ (*finite* ∧ (*empty* ∨ *inf* ∨ (*bi f*)⌢*skip*)) =
(*finite* ∧ (*empty* ∨ (*bi f*)⌢*skip*))
unfolding *finite-d-def* **by** *fastforce*
have *53*: ⊢ (*finite* ∧ (*empty* ∨ (*bi f*)⌢*skip*)) =
(*empty* ∨ (*finite* ∧ (*bi f*)⌢*skip*))
by (*metis* (*no-types*, *opaque-lifting*) *ChopEmpty EmptyOrSChopEqv EmptySChop*
FiniteImportSChopRight int-eq itl-def(*9*))
have *54*: ⊢ (*finite* ∧ (*bi f*)⌢*skip*) = (*bi f*)⌢*skip*
by (*meson Prop10 Prop11 SChopImpFinite WPowerstar-ext WPowerstar-skip-finite lift-and-com lift-imp-trans*)
have *55*: ⊢ (*empty* ∨ (*finite* ∧ (*empty* ∨ *inf* ∨ (*bi f*)⌢*skip*))) =
(*empty* ∨ (*bi f*)⌢*skip*)
using *52 54* **by** *fastforce*
have *6*: ⊢ (*empty* ∨ (*finite* ∧ ¬((*di* (¬ *f*))⌢*skip*))) =
(*empty* ∨ (*bi f*)⌢*skip*)
by (*metis 51 55 inteq-reflection*)
from *2 3 4 6* **show** *?thesis*
by (*simp add*: *ds-d-def*)
qed

lemma *NotBsEqvDsNot*:
⊢ (¬(*bs f*)) = *ds* (¬ *f*)
proof −
have *1*: ⊢ *bs f* = (¬(*ds* (¬ *f*))) **by** (*rule BsEqvNotDsNot*)
hence *2*: ⊢ (¬(*bs f*)) = (¬¬(*ds* (¬ *f*))) **by** *auto*
from *2* **show** *?thesis* **by** *auto*
qed

lemma *NotDsEqvBsNot*:
⊢ (¬ (*ds f*)) = *bs* (¬ *f*)
proof −
have *1*: ⊢ (¬(*ds f*)) = (¬¬(*bs* (¬*f*))) **by** (*simp add*: *ds-d-def*)
from *1* **show** *?thesis* **by** *auto*
qed

lemma *NotDsAndEmpty*:
⊢ ¬(*ds f* ∧ *empty*)
proof −
have *1*: ⊢ *ds f* = (*more* ∧ (*finite* ⟶(*di f*)⌢*skip*)) **by** (*rule DsMoreDi*)
have *2*: ⊢ (*more* ∧ (*finite* ⟶(*di f*)⌢*skip*)) ∧ *empty* ⟶ #*False*
unfolding *empty-d-def* **by** *fastforce*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BsMoreEqvEmpty*:
⊢ *bs more* = *empty*
proof −
have *1*: ⊢ *bs more* = (*empty* ∨ (*bi more*)⌢*skip*)
by (*simp add*: *bs-d-def*)
have *2*: ⊢ *bi more* ⟶ #*False*

by (*metis EmptyChop TrueW bi-d-def di-d-def empty-d-def int-simps*(*14*) *int-simps*(*4*)
inteq-reflection)
hence *3*: ⊢ (*bi more*)⌢*skip* ⟶ #*False*⌢*skip*
using *LeftSChopImpSChop* **by** *blast*
have *31*: ⊢ #*False*⌢*skip* ⟶ #*False*
by (*simp add*: *Valid-def skip-defs schop-defs*)
have *32*: ⊢ (*bi more*)⌢*skip* ⟶ #*False* **using** *3 31* **by** *fastforce*
hence *4*: ⊢ (*empty* ∨ ((*bi more*)⌢*skip*)) = *empty* **by** *fastforce*
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BsAndEqv*:
⊢ (*bs f* ∧ *bs g*) = *bs*(*f* ∧ *g*)
proof −
have *1*: ⊢ *bs f* = (*empty* ∨ (*bi f*) ⌢*skip*)
by (*simp add*: *bs-d-def*)
have *2*: ⊢ *bs g* = (*empty* ∨ (*bi g*) ⌢*skip*)
by (*simp add*: *bs-d-def*)
have *3*: ⊢ (*bs f* ∧ *bs g*) = ((*empty* ∨ (*bi f*) ⌢*skip*) ∧ (*empty* ∨ (*bi g*) ⌢*skip*))
using *1 2* **by** *fastforce*
have *4*: ⊢ ((*empty* ∨ (*bi f*) ⌢*skip*) ∧ (*empty* ∨ (*bi g*) ⌢*skip*)) =
(*empty* ∨ ((*bi f*) ⌢*skip* ∧ (*bi g*) ⌢*skip*))
by *auto*
have *5*: ⊢ (((*bi f*) ⌢*skip* ∧ (*bi g*) ⌢*skip*)) = *bi*(*f* ∧ *g*)⌢*skip*
by (*metis BiAndEqv SChopSkipAndSChopSkip inteq-reflection*)
hence *6*: ⊢ (*empty* ∨ ((*bi f*) ⌢*skip* ∧ (*bi g*) ⌢*skip*)) = (*empty* ∨ *bi*(*f* ∧ *g*)⌢*skip*)
by *auto*
from *3 4 6* **show** *?thesis* **by** (*metis bs-d-def inteq-reflection*)
qed

lemma *BsEqvRule*:
assumes ⊢ *f* = *g*
shows ⊢ *bs f* = *bs g*
proof −
have *1*: ⊢ *f* = *g* **using** *assms* **by** *auto*
hence *2*: ⊢ *bi*(*f*) = *bi*(*g*) **by** (*simp add*: *BiEqvBi*)
hence *3*: ⊢ *bi*(*f*)⌢*skip* = *bi*(*g*)⌢*skip* **by** (*simp add*: *LeftSChopEqvSChop*)
hence *4*: ⊢ (*empty* ∨ *bi*(*f*)⌢*skip*) = (*empty* ∨ *bi*(*g*)⌢*skip*) **by** *auto*
hence *5*: ⊢ *bs*(*f*) = *bs*(*g*) **by** (*simp add*: *bs-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *auto*
qed

lemma *DsEqvRule*:
assumes ⊢ *f* = *g*
shows ⊢ *ds f* = *ds g*
using *assms* **using** *int-eq* **by** *force*

lemma *DsOrEqv*:
⊢ (*ds f* ∨ *ds g*) = *ds* (*f* ∨ *g*)

proof −
have *1*: ⊢ *ds f* = (¬(*bs* (¬ *f*))) **by** (*simp add*: *ds-d-def*)
have *2*: ⊢ *ds g* = (¬(*bs* (¬ *g*))) **by** (*simp add*: *ds-d-def*)
have *3*: ⊢ (*ds f* ∨ *ds g*) = (¬(*bs* (¬ *f*)) ∨ ¬(*bs* (¬ *g*))) **using** *1 2* **by** *fastforce*
have *4*: ⊢ (¬(*bs* (¬ *f*)) ∨ ¬(*bs* (¬ *g*))) = (¬(*bs* (¬ *f*) ∧ *bs* (¬ *g*))) **by** *auto*
have *5*: ⊢ (*bs* (¬ *f*) ∧ *bs* (¬ *g*)) = *bs*(¬ *f* ∧ ¬*g*) **by** (*rule BsAndEqv*)
hence *6*: ⊢ (¬(*bs* (¬ *f*) ∧ *bs* (¬ *g*))) = (¬(*bs*(¬ *f* ∧ ¬*g*))) **by** *auto*
have *7*: ⊢ (¬(*bs*(¬ *f* ∧ ¬*g*))) = *ds* (¬(¬ *f* ∧ ¬*g*)) **by** (*rule NotBsEqvDsNot*)
have *8*: ⊢ (¬(¬ *f* ∧ ¬*g*)) = (*f* ∨ *g*) **by** *auto*
hence *9*: ⊢ *ds*(¬(¬ *f* ∧ ¬*g*)) = *ds* (*f* ∨ *g*) **by** (*rule DsEqvRule*)
from *3 4 6 7 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BsOrImp*:
⊢ *bs f* ∨ *bs g* ⟶ *bs*(*f* ∨ *g*)
proof −
have *1*: ⊢ *bi f* ∨ *bi g* ⟶ *bi*(*f* ∨ *g*)
by (*rule BiOrBiImpBiOr*)
hence *2*: ⊢ (*bi f* ∨ *bi g*)⌢*skip* ⟶ (*bi*(*f* ∨ *g*))⌢*skip*
by (*rule LeftSChopImpSChop*)
have *3*: ⊢ (*bi f*)⌢*skip* ∨ (*bi g*)⌢*skip* ⟶ (*bi*(*f* ∨ *g*))⌢*skip*
using *1 OrSChopEqv 2* **by** *fastforce*
hence *4*: ⊢ *empty* ∨ (*bi f*)⌢*skip* ∨ (*bi g*)⌢*skip* ⟶ *empty* ∨ (*bi*(*f* ∨ *g*))⌢*skip*
by *auto*
hence *5*: ⊢ (*empty* ∨ (*bi f*)⌢*skip*) ∨ (*empty* ∨ (*bi g*)⌢*skip*) ⟶ *empty* ∨ (*bi*(*f* ∨ *g*))⌢*skip*
by *auto*
from *5* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *DsAndImp*:
⊢ *ds* (*f* ∧ *g*) ⟶ *ds f* ∧ *ds g*
proof −
have *1*: ⊢ *bs* (¬*f*) ∨ *bs* (¬*g*) ⟶ *bs*(¬*f* ∨ ¬*g*) **by** (*rule BsOrImp*)
have *2*: ⊢ (¬*f* ∨ ¬*g*) = (¬(*f* ∧ *g*)) **by** *auto*
hence *3*: ⊢ *bs*(¬*f* ∨ ¬*g*) = *bs* (¬(*f* ∧ *g*)) **by** (*rule BsEqvRule*)
have *4*: ⊢ *bs* (¬*f*) ∨ *bs* (¬*g*) ⟶ *bs* (¬(*f* ∧ *g*)) **using** *1 3* **by** *fastforce*
have *5*: ⊢ *bs* (¬*f*) = (¬(*ds f*)) **using** *NotDsEqvBsNot* **by** *fastforce*
have *6*: ⊢ *bs* (¬*g*) = (¬(*ds g*)) **using** *NotDsEqvBsNot* **by** *fastforce*
have *7*: ⊢ *bs* (¬(*f* ∧ *g*)) = (¬(*ds* (*f* ∧ *g*))) **using** *NotDsEqvBsNot* **by** *fastforce*
have *8*: ⊢ ¬(*ds f*) ∨ ¬(*ds g*) ⟶ ¬(*ds* (*f* ∧ *g*)) **using** *4 5 6 7* **by** *fastforce*
hence *9*: ⊢ ¬(*ds f* ∧ *ds g*) ⟶ ¬(*ds* (*f* ∧ *g*)) **by** *auto*
from *9* **show** *?thesis* **by** *auto*
qed

lemma *DsAndImpElimL*:
⊢ *ds* (*f* ∧ *g*) ⟶ *ds f*
using *DsAndImp* **by** *fastforce*

lemma *DsAndImpElimR*:
⊢ *ds* (*f* ∧ *g*) ⟶ *ds g*

using *DsAndImp* **by** *fastforce*

lemma *BiImpBs*:
$\vdash$ *bi f* $\wedge$ *finite* $\longrightarrow$ *bs f*
proof −
have *1*: $\vdash$ *empty* $\longrightarrow$ *empty* $\vee$ (*bi f*)$^\frown$*skip* **by** *auto*
hence *10*: $\vdash$ *empty* $\wedge$ *bi f* $\longrightarrow$ *empty* $\vee$ (*bi f*)$^\frown$*skip* **by** *auto*
have *11*: $\vdash$ *more* $\wedge$ *bi f* $\longrightarrow$ (*f* $\wedge$ *inf*) $\vee$ (*more* $\wedge$ *bi f* $\wedge$ *finite*)
using *BiElim*[*of f*] *OrFiniteInf*[*of LIFT bi f*] **by** *fastforce*
have *12*: $\vdash$ *more* $\wedge$ *bi f* $\wedge$ *finite* $\longrightarrow$ (*bi f*)$^\frown$*skip*
using *MoreAndBiImpBiSChopSkip* **by** *fastforce*
have *2*: $\vdash$ *more* $\wedge$ *bi f* $\longrightarrow$ *inf* $\vee$ (*bi f*)$^\frown$*skip*
using *12* **unfolding** *finite-d-def* **by** *fastforce*
have *3*: $\vdash$ *more* $\wedge$ *bi f* $\wedge$ *finite* $\longrightarrow$ *empty* $\vee$ (*bi f*)$^\frown$*skip*
by (*metis 12 Prop05*)
have *4*: $\vdash$ (*bi f* $\wedge$ *finite*)= ((*bi f* $\wedge$ *empty* $\wedge$ *finite*) $\vee$ (*bi f* $\wedge$ *more* $\wedge$ *finite*))
by (*auto simp add*: *empty-d-def*)
have *41*: $\vdash$ (*bi f* $\wedge$ *empty* $\wedge$ *finite*) = (*bi f* $\wedge$ *empty*)
using *FiniteAndEmptyEqvEmpty* **by** *fastforce*
have *5*: $\vdash$ (*empty* $\vee$ (*bi f*)$^\frown$*skip*) = *bs f* **by** (*simp add*: *bs-d-def*)
have *6*: $\vdash$ (*bi f* $\wedge$ *finite*) = (*bf f* $\wedge$ *finite*)
by (*meson FiniteImpAnd FiniteImpBfEqvBi Prop11*)
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *BfImpBs*:
$\vdash$ *bf f* $\wedge$ *finite* $\longrightarrow$ *bs f*
proof −
have *1*: $\vdash$ *empty* $\longrightarrow$ *empty* $\vee$ (*bi f*)$^\frown$*skip* **by** *auto*
hence *2*: $\vdash$ *empty* $\wedge$ *bi f* $\longrightarrow$ *empty* $\vee$ (*bi f*)$^\frown$*skip* **by** *auto*
have *20*: $\vdash$ *more* $\wedge$ *bi f* $\longrightarrow$ *inf* $\vee$ (*bi f*)$^\frown$*skip*
using *MoreAndBiImpBiSChopSkip* **unfolding** *finite-d-def* **by** *fastforce*
have *21*: $\vdash$ *more* $\wedge$ *bi f* $\wedge$ *finite* $\longrightarrow$ (*bi f*)$^\frown$*skip*
unfolding *finite-d-def* **using** *20* **by** *fastforce*
have *3*: $\vdash$ *more* $\wedge$ *bi f* $\wedge$ *finite* $\longrightarrow$ *empty* $\vee$ (*bi f*)$^\frown$*skip* **using** *21* **by** *auto*
have *4*: $\vdash$ (*bi f* $\wedge$ *finite*) = ((*bi f* $\wedge$ *empty* $\wedge$ *finite*) $\vee$ (*bi f* $\wedge$ *more* $\wedge$ *finite*))
by (*auto simp add*: *empty-d-def*)
have *41*: $\vdash$ (*bi f* $\wedge$ *empty* $\wedge$ *finite*) = (*bi f* $\wedge$ *empty*)
using *FiniteAndEmptyEqvEmpty* **by** *fastforce*
have *5*: $\vdash$ (*empty* $\vee$ (*bi f*)$^\frown$*skip*) = *bs f* **by** (*simp add*: *bs-d-def*)
from *2 3 4 5* **show** *?thesis*
by (*meson BiImpBs FiniteImpAnd FiniteImpBfEqvBi Prop11 lift-imp-trans*)
qed

lemma *BiFiniteEqvFinite*:
$\vdash$ *bi finite* = *finite*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *BsImpBsBs*:
$\vdash$ *bs f* $\longrightarrow$ *bs* (*bs f*)
proof −
have *1*: $\vdash$ *bi f* $\wedge$ *finite* $\longrightarrow$ *bs f*
using *BiImpBs* **by** *auto*
hence *2*: $\vdash$ *bi* (*bi f* $\wedge$ *finite*) $\longrightarrow$ *bi*(*bs f*) **by** (*rule BiImpBiRule*)
have *21*: $\vdash$ *bi* (*bi f* $\wedge$ *finite*) = (*bi f* $\wedge$ *finite*)
by (*metis BiAndEqv BiEqvBiBi BiFiniteEqvFinite inteq-reflection*)
have *3*: $\vdash$ (*bi f*) $\wedge$ *finite* $\longrightarrow$ *bi*(*bs f*) **using** *2 21* **by** *fastforce*
have *4*: $\vdash$ (*bi f*)$\frown$*skip* $\longrightarrow$ (*bi*(*bs f*))$\frown$*skip*
by (*metis* (*no-types*, *opaque-lifting*) *3 LeftChopImpChop Prop12 SChopstar-ext SChopstar-finite lift-imp-trans schop-d-def*)
have *5*: $\vdash$ *empty* $\vee$ (*bi f*)$\frown$*skip* $\longrightarrow$ *empty* $\vee$ (*bi*(*bs f*))$\frown$*skip*
using *4* **by** *fastforce*
have *6*: $\vdash$ (*bi*(*bs f*))$\frown$*skip* = (*bf*(*bs f*))$\frown$*skip*
by (*meson BfEqvImpSChopEqvSChop FiniteBfGen FiniteImpBfEqvBi MP Prop11*)
from *5 6* **show** *?thesis*
by (*metis BsEqvEmptyOrBiSChopSkip inteq-reflection*)
qed

lemma *DsImpDi*:
$\vdash$ *ds f* $\longrightarrow$ *inf* $\vee$ *di f*
proof −
have *1*: $\vdash$ *bi* ($\neg$ *f*) $\wedge$ *finite* $\longrightarrow$ *bs* ($\neg$ *f*) **by** (*rule BiImpBs*)
hence *2*: $\vdash$ $\neg$(*bs* ($\neg$ *f*)) $\longrightarrow$ *inf* $\vee$ $\neg$(*bi* ($\neg$ *f*)) **unfolding** *finite-d-def* **by** *fastforce*
from *2* **show** *?thesis*
using *NotBsEqvDsNot DiEqvNotBiNot* **by** *fastforce*
qed

lemma *DsImpDf*:
$\vdash$ *ds f* $\longrightarrow$ *inf* $\vee$ *df f*
proof −
have *1*: $\vdash$ *bf* ($\neg$ *f*) $\wedge$ *finite* $\longrightarrow$ *bs* ($\neg$ *f*) **by** (*rule BfImpBs*)
hence *2*: $\vdash$ $\neg$(*bs* ($\neg$ *f*)) $\longrightarrow$ *inf* $\vee$$\neg$(*bf* ($\neg$ *f*)) **unfolding** *finite-d-def* **by** *fastforce*
from *2* **show** *?thesis* **by** (*simp add*: *bf-d-def ds-d-def*)
qed

lemma *BsImpBsRule*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ *bs f* $\longrightarrow$ *bs g*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *bi f* $\longrightarrow$ *bi g* **by** (*rule BiImpBiRule*)
hence *3*: $\vdash$ (*bi f*)$\frown$*skip* $\longrightarrow$ (*bi g*)$\frown$*skip* **by** (*rule LeftSChopImpSChop*)
hence *4*: $\vdash$ *empty* $\vee$ (*bi f*)$\frown$*skip* $\longrightarrow$ *empty* $\vee$ (*bi g*)$\frown$*skip* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *DsChopImpDsB*:
$\vdash$ *ds* (*f*;*g*) $\longrightarrow$ *ds f*

proof −
have *1*: ⊢ *di*(*f*;*g*) ⟶ *di f* **by** (*rule DiChopImpDiB*)
hence *2*: ⊢ (*di*(*f*;*g*))⌢*skip* ⟶ (*di f*)⌢*skip* **by** (*rule LeftSChopImpSChop*)
from *2* **show** *?thesis* **using** *DsDi*
by (*metis ChopImpDi DiEqvDiDi DsAndImpElimR Prop10 inteq-reflection*)
qed

lemma *DsSChopImpDsB*:
⊢ *ds* (*f*⌢*g*) ⟶ *ds f*
by (*metis DsAndImpElimL DsChopImpDsB lift-imp-trans schop-d-def*)

lemma *NotBsImpBsNotChop*:
⊢ *bs* (¬ *f*) ⟶ *bs* (¬(*f*;*g*))
proof −
have *1*: ⊢ *ds* (*f*;*g*) ⟶ *ds f* **by** (*rule DsChopImpDsB*)
hence *2*: ⊢ ¬(*ds f*) ⟶ ¬(*ds* (*f*;*g*)) **by** *fastforce*
from *2* **show** *?thesis* **using** *NotDsEqvBsNot* **by** *fastforce*
qed

lemma *BsOrBsEqvBsBiOrBi*:
⊢ (*bs f* ∨ *bs g*) = *bs*(*bi f* ∨ *bi g*)
proof −
have *1*: ⊢ (*bs f* ∨ *bs g*) = ((*empty* ∨ (*bi f*)⌢*skip*) ∨ (*empty* ∨ (*bi g*)⌢*skip*))
by (*simp add*: *bs-d-def*)
have *2*: ⊢ ((*empty* ∨ (*bi f*)⌢*skip*) ∨ (*empty* ∨ (*bi g*)⌢*skip*)) = (*empty* ∨ (*bi f*)⌢*skip* ∨ (*bi g*)⌢*skip*)
by *auto*
have *3*: ⊢ ((*bi f*)⌢*skip* ∨ (*bi g*)⌢*skip*) = (*bi f* ∨ *bi g*)⌢*skip*
by (*meson OrSChopEqv Prop11*)
hence *4*: ⊢ (*empty* ∨ (*bi f*)⌢*skip* ∨ (*bi g*)⌢*skip*) = (*empty* ∨ (*bi f* ∨ *bi g*)⌢*skip*)
by *auto*
have *5*: ⊢ (*bi f* ∨ *bi g*) = *bi* (*bi f* ∨ *bi g*)
by (*meson BiBiOrEqvBi Prop11*)
hence *6*: ⊢ (*bi f* ∨ *bi g*)⌢*skip* = *bi* (*bi f* ∨ *bi g*)⌢*skip*
by (*simp add*: *LeftSChopEqvSChop*)
hence *7*: ⊢ (*empty* ∨ *bi* (*bi f* ∨ *bi g*)⌢*skip*) = (*empty* ∨ (*bi f* ∨ *bi g*)⌢*skip*)
by *auto*
have *8*: ⊢ (*empty* ∨ (*bi f* ∨ *bi g*)⌢*skip*) = *bs*(*bi f* ∨ *bi g*)
by (*metis BiBiOrEqvBi BsEqvEmptyOrBiSChopSkip inteq-reflection*)
from *1 2 4 8* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *BsOrBsEqvBsBfOrBf*:
⊢ (*bs f* ∨ *bs g*) = *bs*(*bf f* ∨ *bf g*)
proof −
have *1*: ⊢ (*bs f* ∨ *bs g*) = *bs*(*bi f* ∨ *bi g*)
by (*simp add*: *BsOrBsEqvBsBiOrBi*)
have *2*: ⊢ (*bi f* ∨ *bi g*)⌢*skip* = (*bf f* ∨ *bf g*)⌢*skip*
by (*metis FiniteImpAnd FiniteImpBfEqvBi OrSChopEqv inteq-reflection schop-d-def*)
from *1 2* **show** *?thesis* **unfolding** *bs-d-def*

by (*metis BfBfOrEqvBf BiBiOrEqvBi FiniteImpAnd FiniteImpBfEqvBi inteq-reflection schop-d-def*)
qed

lemma *DiOrDsEqvDi*:
$\vdash$ *di f* $\vee$ (*ds f* $\wedge$ *finite*) = *di f*
proof −
have *1*: $\vdash$ *di f* $\longrightarrow$ *di f* $\vee$ *ds f* **by** *auto*
have *2*: $\vdash$ *di f* $\longrightarrow$ *di f* **by** *auto*
have *3*: $\vdash$ *ds f* $\longrightarrow$ *inf* $\vee$ *di f* **by** (*rule DsImpDi*)
have *4*: $\vdash$ *di f* $\vee$ *ds f* $\longrightarrow$ *inf* $\vee$ *di f* **using** *2 3* **by** *auto*
from *1 4* **show** *?thesis* **unfolding** *finite-d-def* **by** *fastforce*
qed

lemma *DiAndDsEqvDs*:
$\vdash$ (*di f* $\wedge$ (*ds f* $\wedge$ *finite*)) = (*ds f* $\wedge$ *finite*)
proof −
have *1*: $\vdash$ *di f* $\wedge$ *ds f* $\longrightarrow$ *ds f* **by** *auto*
have *2*: $\vdash$ *ds f* $\longrightarrow$ *ds f* **by** *auto*
have *3*: $\vdash$ *ds f* $\longrightarrow$ *inf* $\vee$ *di f* **by** (*rule DsImpDi*)
have *4*: $\vdash$ (*ds f* $\wedge$ *finite*)$\longrightarrow$ *di f* $\wedge$ (*ds f* $\wedge$ *finite*)
using *2 3* **unfolding** *finite-d-def* **by** *auto*
from *1 4* **show** *?thesis* **by** *auto*
qed

lemma *DiFiniteOrInf*:
$\vdash$ *di f* = ((*f* $\wedge$ *finite*) $\vee$ (*di f*)$\frown$*skip* $\vee$ (*di f* $\wedge$ *inf*))
proof −
have *1*: $\vdash$ (*di f*) = ((*di f* $\wedge$ *finite*) $\vee$ (*di f* $\wedge$ *inf*))
by (*simp add*: *OrFiniteInf*)
have *2*: $\vdash$ (*di f* $\wedge$ *finite*) = (*di f*)$\frown$*finite*
by (*metis ChopTrueAndFiniteEqvAndFiniteChopFinite DiEqvDiDi di-d-def inteq-reflection schop-d-def*)
have *3*: $\vdash$ (*more* $\wedge$ *finite*) = (#*True*$\frown$*skip*)
by (*metis DiamondSChopdef FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip fmore-d-def inteq-reflection sometimes-d-def*)
have *4*: $\vdash$ *finite* = (#*True*$\frown$*skip* $\vee$ *empty*)
using *3* **unfolding** *empty-d-def*
by (*metis DiamondEmptyEqvFinite FmoreEqvSkipChopFinite WPowerstarChopEqvChopOrRule WPowerstar-skip-finite empty-d-def fmore-d-def int-eq sometimes-d-def*)
have *5*: $\vdash$ (*di f*)$\frown$*finite* = (*di f*)$\frown$(#*True*$\frown$*skip* $\vee$ *empty*)
by (*simp add*: *4 RightSChopEqvSChop*)
have *6*: $\vdash$ (*di f*)$\frown$(#*True*$\frown$*skip*) = (*di f*)$\frown$*skip*
unfolding *di-d-def*
by (*metis 2 3 ChopAssoc FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip di-d-def fmore-d-def inteq-reflection schop-d-def*)
have *7*: $\vdash$ (*di f*) $\frown$ *finite* = (*f* $\frown$ *finite*)$\frown$*finite*
by (*metis* (*no-types*, *lifting*) *4 ChopTrueAndFiniteEqvAndFiniteChopFinite Prop10 Prop11 SChopImpFinite di-d-def inteq-reflection schop-d-def*)
have *8*: $\vdash$ (*f* $\frown$ *finite*)$\frown$*finite* = (*f* $\frown$ *finite*)

by (*metis 2 7 ChopTrueAndFiniteEqvAndFiniteChopFinite Prop04 di-d-def schop-d-def*)
have *9*: $\vdash (f \frown \mathit{finite}) = f \frown (\#\mathit{True} \frown \mathit{skip} \vee \mathit{empty})$
by (*simp add*: *4 RightSChopEqvSChop*)
have *10*: $\vdash f \frown (\#\mathit{True} \frown \mathit{skip}) = (\mathit{di}\ f) \frown \mathit{skip}$
by (*metis* (*no-types*, *opaque-lifting*) *3 ChopAssoc ChopTrueAndFiniteEqvAndFiniteChopFinite FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip di-d-def fmore-d-def inteq-reflection schop-d-def*)
have *11*: $\vdash f \frown \mathit{empty} = (f \wedge \mathit{finite})$
by (*simp add*: *ChopEmpty schop-d-def*)
have *12*: $\vdash f \frown (\#\mathit{True} \frown \mathit{skip} \vee \mathit{empty}) = (f \frown (\#\mathit{True} \frown \mathit{skip}) \vee f \frown \mathit{empty})$
by (*simp add*: *SChopOrEqv*)
have *13*: $\vdash (\mathit{di}\ f \wedge \mathit{finite}) = ((f \wedge \mathit{finite}) \vee (\mathit{di}\ f) \frown \mathit{skip})$
by (*metis 10 11 2 3 7 8 FmoreEqvSkipChopFinite SChopOrEqvRule WPowerstarEqv WPowerstar-skip-finite fmore-d-def int-eq*)
have *14*: $\vdash ((\mathit{di}\ f \wedge \mathit{finite}) \vee (\mathit{di}\ f \wedge \mathit{inf})) = ((f \wedge \mathit{finite}) \vee (\mathit{di}\ f) \frown \mathit{skip} \vee (\mathit{di}\ f \wedge \mathit{inf}))$
using *13* **by** *fastforce*
show *?thesis* **using** *1 14* **by** *fastforce*
qed

lemma *DiFiniteEqv*:
$\vdash (\mathit{di}\ f \wedge \mathit{finite}) = ((f \wedge \mathit{finite}) \vee (\mathit{di}\ f) \frown \mathit{skip})$
proof −
have *1*: $\vdash (\mathit{di}\ f \wedge \mathit{finite}) = (((f \wedge \mathit{finite}) \vee (\mathit{di}\ f) \frown \mathit{skip} \vee (\mathit{di}\ f \wedge \mathit{inf}))\ \wedge \mathit{finite})$
using *DiFiniteOrInf* **by** (*metis int-simps*(*1*) *inteq-reflection*)
have *2*: $\vdash (((f \wedge \mathit{finite}) \vee (\mathit{di}\ f) \frown \mathit{skip} \vee (\mathit{di}\ f \wedge \mathit{inf}))\ \wedge \mathit{finite}) = ((f \wedge \mathit{finite}) \vee ((\mathit{di}\ f) \frown \mathit{skip} \wedge \mathit{finite}))$
by (*metis* (*no-types*, *opaque-lifting*) *ChopAndFiniteDist ChopEmpty DiEqvOrDiChopSkipB DiFiniteOrInf OrSChopEqv SChopAssoc inteq-reflection itl-def*(*9*))
have *3*: $\vdash ((\mathit{di}\ f) \frown \mathit{skip} \wedge \mathit{finite}) = ((\mathit{di}\ f) \frown \mathit{skip})$
by (*metis Prop10 SChopImpFinite WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
show *?thesis*
by (*metis 1 2 3 inteq-reflection*)
qed

lemma *OrDsEqvDi*:
$\vdash ((f \wedge \mathit{finite}) \vee (\mathit{ds}\ f \wedge \mathit{finite})) = (\mathit{di}\ f \wedge \mathit{finite})$
proof −
have *1*: $\vdash (\mathit{ds}\ f \wedge \mathit{finite}) = ((\mathit{finite} \longrightarrow (\mathit{di}\ f) \frown \mathit{skip}) \wedge \mathit{finite})$
using *DsDi* **by** *fastforce*
have *11*: $\vdash ((\mathit{finite} \longrightarrow (\mathit{di}\ f) \frown \mathit{skip}) \wedge \mathit{finite}) = (((\mathit{di}\ f) \frown \mathit{skip}) \wedge \mathit{finite})$
by *fastforce*
have *12*: $\vdash (((\mathit{di}\ f) \frown \mathit{skip}) \wedge \mathit{finite}) = (\mathit{di}\ f) \frown \mathit{skip}$
by (*metis Prop10 SChopImpFinite WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
have *2*: $\vdash ((f \wedge \mathit{finite}) \vee (\mathit{ds}\ f \wedge \mathit{finite})) = ((f \wedge \mathit{finite}) \vee (\mathit{di}\ f) \frown \mathit{skip})$
by (*metis 1 11 12 DiFiniteEqv inteq-reflection*)
have *3*: $\vdash ((f \wedge \mathit{finite}) \vee (\mathit{di}\ f) \frown \mathit{skip}) = (\mathit{di}\ f \wedge \mathit{finite})$
by (*meson DiFiniteEqv Prop11*)

from *2 3* **show** *?thesis*
by (*metis int-eq*)
qed

lemma *BiFiniteEqv*:
$\vdash (bi\ f \wedge finite) = ((f \wedge finite) \wedge (empty \vee (bi\ f) \frown skip))$
by (*metis* (*no-types*, *opaque-lifting*) *BiAndEqv BiEqvAndWprevBi BiFiniteEqvFinite WPrevAndFiniteEqv inteq-reflection*)

lemma *AndSChopSkipEqVFiniteAndSChopSkip*:
$\vdash (f \wedge g \frown skip) = ((f \wedge finite) \wedge g \frown skip)$
proof −
have *1*: $\vdash g \frown skip \longrightarrow finite$
by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)
from *1* **show** *?thesis*
by *fastforce*
qed

lemma *BiFiniteEQvB*:
$\vdash (bi\ f \wedge finite) = (f \wedge (empty \vee (bi\ f) \frown skip))$
proof −
have *1*: $\vdash ((f \wedge finite) \wedge empty) = (f \wedge empty)$
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have *2*: $\vdash ((f \wedge finite) \wedge (bi\ f) \frown skip) = (f \wedge (bi\ f) \frown skip)$
by (*meson AndSChopSkipEqVFiniteAndSChopSkip Prop11*)
have *4*: $\vdash (((f \wedge finite) \wedge empty) \vee ((f \wedge finite) \wedge (bi\ f) \frown skip)) =$
$((f \wedge empty) \vee (f \wedge (bi\ f) \frown skip))$
using *1 2* **by** *fastforce*
have *3*: $\vdash (bi\ f \wedge finite) = ((f \wedge finite) \wedge (empty \vee (bi\ f) \frown skip))$
by (*simp add*: *BiFiniteEqv*)
show *?thesis* **using** *3 4* **by** *fastforce*
qed

lemma *AndBsEqvBi*:
$\vdash (f \wedge bs\ f) = (bi\ f \wedge finite)$
by (*metis BiFiniteEQvB BsEqvEmptyOrBiSChopSkip inteq-reflection*)

lemma *BsEqvBsBi*:
$\vdash bs\ f = bs\ (bi\ f)$
proof −
have *1*: $\vdash bs\ f = (empty \vee (bi\ f) \frown skip)$ **by** (*simp add*: *bs-d-def*)
have *2*: $\vdash bi\ f = bi\ (\ bi\ f)$ **by** (*rule BiEqvBiBi*)
hence *3*: $\vdash (bi\ f) \frown skip = bi\ (bi\ f) \frown skip$ **using** *LeftSChopEqvSChop* **by** *blast*
hence *4*: $\vdash (empty \vee (bi\ f) \frown skip) = (empty \vee bi\ (bi\ f) \frown skip)$ **by** *auto*
from *1 4* **show** *?thesis* **unfolding** *bs-d-def* **by** *auto*
qed

lemma *StateImpBs*:
$\vdash$ *init w* $\wedge$ *finite* $\longrightarrow$ *bs* (*init w*)
proof −
have *1*: $\vdash$ *init w* = *bi* (*init w*) **by** (*rule StateEqvBi*)
have *2*: $\vdash$ *bi* (*init w*) $\wedge$ *finite* $\longrightarrow$ *bs* (*init w*) **by** (*rule BiImpBs*)
from *1 2* **show** *?thesis* **using** *StateImpBi* **by** *fastforce*
qed

lemma *DsAndDsEqvDsAndDiOrDsAndDi*:
$\vdash$ (*ds f* $\wedge$ *ds g*) =(*ds* (*f* $\wedge$ *di g*) $\vee$ *ds*(*g* $\wedge$ *di f*))
proof −
have *1*: $\vdash$ (*di f* $\wedge$ *di g*) = (*di*(*f* $\wedge$ *di g*) $\vee$ *di*(*g* $\wedge$ *di f*))
by (*rule DiAndDiEqvDiAndDiOrDiAndDi*)
hence *2*: $\vdash$ (*di f* $\wedge$ *di g*)$\frown$*skip* = (*di*(*f* $\wedge$ *di g*) $\vee$ *di*(*g* $\wedge$ *di f*))$\frown$*skip*
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash$ (*di f* $\wedge$ *di g*)$\frown$*skip* = ((*di f*)$\frown$*skip* $\wedge$ (*di g*)$\frown$*skip*)
using *SChopSkipAndSChopSkip* **by** *fastforce*
have *4*: $\vdash$ ((*di f*)$\frown$*skip* $\wedge$ (*di g*)$\frown$*skip*) = (*di*(*f* $\wedge$ *di g*) $\vee$ *di*(*g* $\wedge$ *di f*))$\frown$*skip*
using *2 3* **by** *fastforce*
have *5*: $\vdash$ (*di*(*f* $\wedge$ *di g*) $\vee$ *di*(*g* $\wedge$ *di f*))$\frown$*skip* = (*di*(*f* $\wedge$ *di g*)$\frown$*skip* $\vee$ *di*(*g* $\wedge$ *di f*)$\frown$*skip*)
using *OrSChopEqv* **by** *blast*
have *6*: $\vdash$ *ds f* = (*finite* $\longrightarrow$(*di f*)$\frown$*skip*)
using *DsDi* **by** *blast*
have *7*: $\vdash$ *ds g* = (*finite* $\longrightarrow$(*di g*)$\frown$*skip*)
using *DsDi* **by** *blast*
have *8*: $\vdash$ ((*finite*$\longrightarrow$(*di f*)$\frown$*skip*) $\wedge$ (*finite* $\longrightarrow$(*di g*)$\frown$*skip*)) = (*ds f* $\wedge$ *ds g*)
using *6 7* **by** *fastforce*
have *81*: $\vdash$ ((*finite* $\longrightarrow$ *di f* $\frown$ *skip*) $\wedge$ (*finite* $\longrightarrow$ *di g* $\frown$ *skip*)) =
(*finite* $\longrightarrow$ *di f* $\frown$ *skip* $\wedge$ *di g* $\frown$ *skip*)
by *fastforce*
have *9*: $\vdash$ *ds*(*f* $\wedge$ *di g*) = (*finite* $\longrightarrow$ *di*(*f* $\wedge$ *di g*)$\frown$*skip*)
using *DsDi* **by** *blast*
have *10*: $\vdash$ *ds*(*g* $\wedge$ *di f*) = (*finite* $\longrightarrow$*di*(*g* $\wedge$ *di f*)$\frown$*skip*)
using *DsDi* **by** *blast*
have *11*: $\vdash$ ((*finite* $\longrightarrow$ *di*(*f* $\wedge$ *di g*)$\frown$*skip*) $\vee$ (*finite* $\longrightarrow$ *di*(*g* $\wedge$ *di f*)$\frown$*skip*)) =
(*ds*(*f* $\wedge$ *di g*) $\vee$ *ds*(*g* $\wedge$ *di f*))
using *9 10* **by** *fastforce*
have *12*: $\vdash$ ((*finite* $\longrightarrow$ *di*(*f* $\wedge$ *di g*)$\frown$*skip*) $\vee$ (*finite* $\longrightarrow$ *di*(*g* $\wedge$ *di f*)$\frown$*skip*)) =
(*finite* $\longrightarrow$ *di*(*f* $\wedge$ *di g*)$\frown$*skip* $\vee$ *di*(*g* $\wedge$ *di f*)$\frown$*skip*)
by *fastforce*
from *4 5 8 11 12 81* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SChopFiniteEqvSChopTrueAndFinite*:
$\vdash$ *g*$\frown$*finite* = (*g*$\frown$#*True* $\wedge$ *finite*)
by (*meson FiniteImportSChopRight Prop04 RightSChopEqvSChop int-simps*(*17*) *lift-and-com*)

lemma *BsEqvBiMoreImpChop*:
$\vdash$ (*bs f*) = (*bi*(*more* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$*skip*) $\wedge$ *finite*)
proof −

have *1*: $\vdash$ (*bs f* $\vee$ *inf*) = (*empty* $\vee$ *inf* $\vee$ (*bi f*$\frown$*skip*))
unfolding *bs-d-def* **by** *fastforce*
have *2*: $\vdash$ (*empty* $\vee$ *inf* $\vee$ (*bi f*$\frown$*skip*)) = ((¬(¬(*bi f*))$\frown$*skip*))
using *NotNotSChopSkip* **by** *fastforce*
have *3*: $\vdash$ ¬((¬(*bi f*))$\frown$*skip*) = (¬(*di* (¬ *f*)$\frown$*skip*))
by (*simp add*: *bi-d-def*)
have *4*: $\vdash$ (¬(*di* (¬ *f*)$\frown$*skip*)) = (¬(((¬ *f*) $\frown$#*True*)$\frown$*skip*))
by (*metis FiniteImpAnd FiniteImpDfEqvDi LeftChopImpChop Prop11 itl-def*(*15*) *lift-imp-neg schop-d-def*)
have *5*: $\vdash$ (¬(((¬ *f*) $\frown$#*True*)$\frown$*skip*)) = (¬((¬ *f*) $\frown$(#*True*$\frown$*skip*)))
by (*meson Prop11 SChopAssoc lift-imp-neg*)
have *05*: $\vdash$ #*True*$\frown$*skip* = *skip* $\frown$ *finite*
by (*metis DiamondSChopdef Prop10 SkipFiniteEqvFiniteSkip WPowerstar-ext WPowerstar-skip-finite inteq-reflection itl-def*(*10*) *itl-def*(*9*))
have *6*: $\vdash$ (¬((¬ *f*) $\frown$(#*True*$\frown$*skip*))) = (¬((¬ *f*) $\frown$(*skip*$\frown$*finite*)))
by (*metis 05 5 inteq-reflection*)
have *7*: $\vdash$ (¬((¬ *f*) $\frown$(*skip*$\frown$*finite*))) = (¬(((¬ *f*) $\frown$*skip*)$\frown$*finite*))
using *SChopAssoc* **by** *fastforce*
have *07*: $\vdash$ (¬(((¬ *f*) $\frown$*skip*)$\frown$*finite*)) = ((¬(((¬ *f*) $\frown$*skip*)$\frown$#*True*)) $\vee$ *inf*)
using *InfEqvNotFinite SChopFiniteEqvSChopTrueAndFinite* **by** *fastforce*
have *8*: $\vdash$ (¬(((¬ *f*) $\frown$*skip*)$\frown$*finite*)) = ((¬(*di* ((¬ *f*)$\frown$*skip*))) $\vee$ *inf*)
by (*metis 07 FiniteImportSChopRight Prop10 WPowerstar-ext WPowerstar-skip-finite di-d-def inteq-reflection lift-and-com schop-d-def*)
have *9*: $\vdash$ (¬(*di* ((¬ *f*)$\frown$*skip*))) = *bi* (¬((¬ *f*) $\frown$*skip*))
using *NotDiEqvBiNot* **by** *blast*
have *10*: $\vdash$ *bi* (¬((¬ *f*) $\frown$*skip*)) = *bi*(*empty* $\vee$ *inf* $\vee$ (*f*$\frown$*skip*))
using *NotNotSChopSkip* **using** *BiEqvBi* **by** *blast*
have *010*: $\vdash$ (*empty* $\vee$ *inf* $\vee$ (*f*$\frown$*skip*)) = (¬ (*more* $\wedge$ *finite*) $\vee$ (*f*$\frown$*skip*))
unfolding *empty-d-def finite-d-def* **by** *fastforce*
have *11*: $\vdash$ *bi*(*empty* $\vee$ *inf* $\vee$ (*f*$\frown$*skip*)) = *bi*(¬ (*more* $\wedge$ *finite*) $\vee$ (*f*$\frown$*skip*))
by (*simp add*: *010 BiEqvBi*)
have *12*: $\vdash$ (¬ (*more* $\wedge$ *finite*) $\vee$ (*f*$\frown$*skip*)) = (*more* $\wedge$ *finite*$\longrightarrow$ *f*$\frown$*skip*) **by** *auto*
have *13*: $\vdash$ *bi*(¬ (*more* $\wedge$ *finite*) $\vee$ (*f*$\frown$*skip*)) = *bi*(*more* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$*skip*)
using *12* **using** *BiEqvBi* **by** *blast*
have *14*: $\vdash$ (*bs f* $\vee$ *inf*) = (¬ (((¬ *f*)$\frown$*skip*)$\frown$*finite*)) **using** *1 2 3 4 5 6 7*
by (*metis NotSChopNotSkip bi-d-def inteq-reflection*)
have *15*: $\vdash$ (¬ (((¬ *f*)$\frown$*skip*)$\frown$*finite*)) = (*bi*(*more* $\wedge$ *finite*$\longrightarrow$ *f*$\frown$*skip*) $\vee$ *inf*)
using *8 9 10 11 13* **by** (*metis inteq-reflection*)
have *16*: $\vdash$ (*bs f* $\vee$ *inf*) = (*bi*(*more* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$*skip*) $\vee$ *inf*)
using *14 15* **by** *fastforce*
have *17*: $\vdash$ *bs f* $\longrightarrow$ *finite*
by (*metis AndChopB EmptyImpFinite FiniteChopSkipImpFinite Prop02 bs-d-def lift-imp-trans schop-d-def*)
have *18*: $\vdash$ ((*bs f* $\vee$ *inf*) $\wedge$ *finite*) = *bs f*
using *17* **unfolding** *finite-d-def* **by** *fastforce*
have *19*: $\vdash$ ((*bi*(*more* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$*skip*) $\vee$ *inf*) $\wedge$ *finite*) =
(*bi*(*more* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$*skip*) $\wedge$ *finite*)
unfolding *finite-d-def* **by** *fastforce*
have *20*: $\vdash$ ((*bs f* $\vee$ *inf*) $\wedge$ *finite*) = ((*bi*(*more* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$*skip*) $\vee$ *inf*) $\wedge$ *finite*)
using *16* **by** *fastforce*
show *?thesis* **using** *16 18 19 20* **by** (*metis inteq-reflection*)
qed

lemma *NotSChopInf*:
$\vdash (\neg\ (\ f \frown inf)) = (finite \vee \neg(f \frown \#True))$
proof −
have *1*: $\vdash (\neg\ (\ f \frown inf)) = (finite \vee (bf\ (\neg\ f)))$
by (*metis NotNotSChopInf int-simps*(*4*) *inteq-reflection*)
have *2*: $\vdash bf\ (\neg\ f) = (\neg\ (f \frown \#True))$
unfolding *bf-d-def df-d-def* **by** *auto*
show *?thesis* **using** *1 2 Prop06* **by** *blast*
qed

lemma *BoxMoreStateEqvBsFinState*:
$\vdash (\Box(more \longrightarrow \neg\ (init\ w)) \wedge finite) = bs(\neg(fin(init\ w)))$
proof −
have *1*: $\vdash \Box(more \longrightarrow \neg\ (init\ w)) = (\neg(\Diamond(\neg(more \longrightarrow \neg\ (init\ w)))))$
by (*simp add*: *always-d-def*)
have *01*: $\vdash (\neg(more \longrightarrow \neg\ (init\ w))) = (init\ w \wedge\ more)$ **by** *auto*
hence *2*: $\vdash \neg(\Diamond(\neg(more \longrightarrow \neg\ (init\ w)))) = (\neg(\#True \frown (init\ w \wedge\ more)))$
by (*metis DiamondSChopdef TrueW int-simps*(*3*) *int-simps*(*6*) *inteq-reflection*)
have *3*: $\vdash more = (\#True \frown\ skip \vee inf)$
by (*metis FmoreEqvSkipChopFinite MoreAndInfEqvInf OrFiniteInf SkipFiniteEqvFiniteSkip*
fmore-d-def int-simps(*17*) *inteq-reflection schop-d-def*)
have *4*: $\vdash (init\ w \wedge\ more) = (init\ w \wedge ((\#True \frown\ skip \vee inf)))$
using *3* **by** *auto*
have *5*: $\vdash (init\ w \wedge (\#True \frown\ skip \vee inf)) \ =$
$((init\ w \wedge empty) \frown (\#True \frown skip) \vee (init\ w \wedge empty) \frown inf)$
by (*meson Prop04 SChopOrEqv StateAndEmptySChop*)
have *6*: $\vdash (init\ w \wedge\ more) = ((init\ w \wedge empty) \frown (\#True \frown skip) \vee (init\ w \wedge empty) \frown inf)$
using *4 5* **by** *fastforce*
have *7*: $\vdash\ (\#True \frown (init\ w \wedge\ more)) =$
$(\#True \frown ((init\ w \wedge empty) \frown (\#True \frown skip) \vee (init\ w \wedge empty) \frown inf)\ )$
using *6* **by** (*simp add*: *RightSChopEqvSChop*)
have *8*: $\vdash (\#True \frown ((init\ w \wedge empty) \frown (\#True \frown skip))\ ) =$
$(((\#True \frown (init\ w \wedge empty)) \frown (\#True \frown skip))\ )$
using *SChopAssoc* **by** *blast*
have *9*: $\vdash (((\#True \frown (init\ w \wedge empty)) \frown (\#True \frown skip))\ ) =$
$((((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip)\ )$
using *SChopAssoc* **by** *blast*
have *10*: $\vdash (\#True \frown (init\ w \wedge\ more)) =$
$((((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip) \vee \#True \frown ((init\ w \wedge empty) \frown inf)\ )$
using *7 8 9* **by** (*metis SChopOrEqv inteq-reflection*)
hence *11*: $\vdash (\neg(\#True \frown (init\ w \wedge\ more))) =$
$(\neg((((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip) \vee \#True \frown ((init\ w \wedge empty) \frown inf)\ ))$
by *auto*
have *011*: $\vdash (\neg((((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip) \vee \#True \frown ((init\ w \wedge empty) \frown inf)\ )) =$
$(\neg((((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip)\ ) \wedge \neg(\ \#True \frown ((init\ w \wedge empty) \frown inf)))$
by *fastforce*
have *12*: $\vdash (\neg((((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip)\ )) =$
$(empty \vee inf \vee (\neg((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip))$
using *NotSChopNotSkip* **by** *fastforce*

have *012*: $\vdash (\neg(\ \#True \frown ((init\ w \wedge empty) \frown inf))) = (finite \vee (\neg((\#True \frown (init\ w \wedge empty))) \frown \#True))$

using *NotSChopInf SChopAssoc* **by** *fastforce*
have *0130*: $\vdash (finite;(init\ w \wedge empty)) = finite;\ (w \wedge empty)$
by (*simp add*: *InitAndEmptyEqvAndEmpty RightChopEqvChop*)
have *0131*: $\vdash ((init\ w \wedge empty) \wedge finite) = (w \wedge empty)$
by (*metis ChopEmpty InitAndEmptyEqvAndEmpty StateAndEmptySChop inteq-reflection schop-d-def*)
have *o132*: $\vdash \#True \frown ((init\ w \wedge empty) \wedge finite) = finite;(w \wedge empty)$
by (*simp add*: *0131 ChopEqvChop schop-d-def*)
have *013*: $\vdash (finite;(init\ w \wedge empty)) = \#True \frown ((init\ w \wedge empty) \wedge finite)$
using *0130 o132* **by** *fastforce*
have *13*: $\vdash (\neg((\#True \frown (init\ w \wedge empty)) \frown \#True)) =$
$bi(\Box\ (\neg(init\ w \wedge empty)))$
using *BiBoxNotEqvNotFiniteChopChopTrue*[*of LIFT* $((init\ w \wedge empty))$]
by (*metis* (*no-types*, *lifting*) *013 ChopAssoc SChopAssoc inteq-reflection schop-d-def*)
hence *14*: $\vdash (\neg((\#True \frown (init\ w \wedge empty)) \frown \#True)) \frown skip =$
$(bi(\Box\ (\neg(init\ w \wedge empty)))) \frown skip$ **using** *LeftSChopEqvSChop* **by** *blast*
hence *15*: $\vdash (empty \vee inf \vee (\neg((\#True \frown (init\ w \wedge empty)) \frown \#True)) \frown skip) =$
$(empty \vee inf \vee (bi(\Box\ (\neg(init\ w \wedge empty)))) \frown skip)$
by *auto*
have *015*: $\vdash (finite \vee (\neg((\#True \frown (init\ w \wedge empty))) \frown \#True)) =$
$(finite \vee bi(\Box\ (\neg(init\ w \wedge empty))))$
using *13* **by** *auto*
have *16*: $\vdash (\neg((((\#True \frown (init\ w \wedge empty)) \frown \#True) \frown skip) \vee \#True \frown ((init\ w \wedge empty) \frown inf)\ )) =$
$((empty \vee inf \vee (bi(\Box\ (\neg(init\ w \wedge empty))) \frown skip)) \wedge (finite \vee bi(\Box\ (\neg(init\ w \wedge empty)))))$
by (*metis 011 012 13 NotSChopNotSkip inteq-reflection*)
have *171*: $\vdash (\neg(init\ w \wedge empty)) = (\neg(init\ w) \vee \neg empty)$
by *auto*
hence *172*: $\vdash \Box(\neg(init\ w \wedge empty)) = \Box(\neg(init\ w) \vee \neg empty)$
by (*simp add*: *BoxEqvBox*)
hence *173*: $\vdash bi(\Box(\neg(init\ w \wedge empty))) = bi(\Box(\neg(init\ w) \vee \neg empty))$
by (*simp add*: *BiEqvBi*)
hence *174*: $\vdash bi(\Box(\neg(init\ w \wedge empty))) \frown skip = bi(\Box(\neg(init\ w) \vee \neg empty)) \frown skip$
using *LeftSChopEqvSChop* **by** *blast*
hence *17*: $\vdash (empty \vee inf \vee (bi(\Box\ (\neg(init\ w \wedge empty))) \frown skip)) =$
$(empty \vee inf \vee (bi(\Box\ (\neg(init\ w) \vee \neg empty)) \frown skip))$
by *auto*
have *175*: $\vdash (finite \vee bi(\Box\ (\neg(init\ w \wedge empty)))) =$
$(finite \vee bi(\Box\ (\neg(init\ w) \vee \neg empty)))$
using *173* **by** *fastforce*
have *181*: $\vdash\ (\neg(init\ w) \vee \neg empty) =\ (\neg empty \vee \neg(init\ w)\ \ )$
by *auto*
hence *18*: $\vdash \Box\ (\neg(init\ w) \vee \neg empty) = \Box\ (\neg empty \vee \neg(init\ w)\ \ )$
by (*simp add*: *BoxEqvBox*)
have *191*: $\vdash\ (\neg empty \vee \neg(init\ w)\ \ ) = (empty \longrightarrow \neg(init\ w)\ )$
by *auto*
hence *19*: $\vdash \Box\ (\neg empty \vee \neg(init\ w)\ \ ) = \Box(empty \longrightarrow \neg(init\ w)\ )$
by (*simp add*: *BoxEqvBox*)
have *20*: $\vdash \Box(empty \longrightarrow \neg(init\ w)\ ) = fin\ (\neg(init\ w)\ )$
by (*simp add*: *fin-d-def*)

have *21*: $\vdash$ (*fin* (¬(*init w*)) ∧ *finite*) = ((¬(*fin* (*init w*)) ∧ *finite*))
by (*metis FinNotStateEqvNotFinState Initprop*(*2*) *inteq-reflection*)
have *22*: $\vdash$ (*bi*(□ (¬(*init w*) ∨ ¬*empty*)) ∧ *finite*) = (*bi* (¬(*fin* (*init w*))) ∧ *finite*)
by (*metis 181 191 20 21 BiAndEqv BiFiniteEqvFinite inteq-reflection*)
hence *23*: $\vdash$ (*bi*(□ (¬(*init w*) ∨ ¬*empty*)))⌢*skip* = (*bi* (¬(*fin* (*init w*))))⌢*skip*
by (*simp add*: *LeftChopEqvChop schop-d-def*)
hence *24*: $\vdash$ (*empty* ∨ *inf* ∨ (*bi*(□ (¬(*init w*) ∨ ¬*empty*)))⌢*skip*) =
(*empty* ∨ *inf* ∨ (*bi* (¬(*fin* (*init w*))))⌢*skip*)
by *auto*
have *241*:$\vdash$ (*bi* (¬(*fin* (*init w*))))⌢*skip* = *bf* (¬ *fin* (*init w*)) ⌢ *skip*
by (*metis* (*no-types*, *opaque-lifting*) *EmptyChop FiniteImpAnd FiniteImpBfEqvBi LeftChopEqvChop Prop04 schop-d-def*)
hence *25*: $\vdash$ (*empty* ∨ *inf* ∨ (*bi* (¬(*fin* (*init w*))))⌢*skip*) = (*inf* ∨ *bs*(¬(*fin* (*init w*))))
unfolding *bs-d-def* **by** *auto*
have *26*:$\vdash$ (□(*more* ⟶ ¬ (*init w*)) ∧ *finite*) =
(((*empty* ∨ *inf* ∨(*bi*(□ (¬(*init w* ∧ *empty*)))⌢*skip*)) ∧ (*finite* ∨ *bi*(□ (¬(*init w* ∧ *empty*)))))
∧ *finite*)
by (*metis* (*no-types*, *opaque-lifting*) *01 10 16 TrueSChopEqvDiamond always-d-def int-simps*(*4*) *inteq-reflection*)
have *27*: $\vdash$ (((*empty* ∨ *inf* ∨(*bi*(□ (¬(*init w* ∧ *empty*)))⌢*skip*)) ∧ (*finite* ∨ *bi*(□ (¬(*init w* ∧ *empty*)))))
∧ *finite*) =
(((*empty* ∨ *inf* ∨(*bi*(□ (¬(*init w* ∧ *empty*)))⌢*skip*)))
∧ *finite*)
by *fastforce*
have *28*: $\vdash$ (*bi*(□ (¬(*init w* ∧ *empty*)))⌢*skip*) ⟶ *finite*
by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)
have *29*:$\vdash$ *empty* ⟶ *finite*
by (*simp add*: *EmptyImpFinite*)
have *29*: $\vdash$ (((*empty* ∨ *inf* ∨(*bi*(□ (¬(*init w* ∧ *empty*)))⌢*skip*))) ∧ *finite*) =
(((*empty* ∨ (*bi*(□ (¬(*init w* ∧ *empty*)))⌢*skip*))))
using *28 29* **unfolding** *finite-d-def* **by** *fastforce*
have *30*: $\vdash$ (((*empty* ∨ (*bi*(□ (¬(*init w* ∧ *empty*)))⌢*skip*)))) = *bs*(¬(*fin* (*init w*)))
by (*metis 171 23 241 29 bs-d-def inteq-reflection*)
show *?thesis*
by (*metis 26 27 29 30 inteq-reflection*)
qed

lemma *BsFalseEqvEmpty*:
$\vdash$ *bs #False* = *empty*
proof −
have *1*: $\vdash$ *bs #False* = (*empty* ∨ *bi #False*⌢*skip*)
by (*simp add*: *bs-d-def*)
have *2*: $\vdash$ ¬(*bi #False*⌢*skip*)
by (*metis BiFiniteEQvB MoreEqvSkipChopTrue NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip
TrueW int-simps(*19*) *int-simps*(*2*) *int-simps*(*21*) *int-simps*(*3*) *inteq-reflection schop-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

First occurrence

lemma *FstEqvRule*:
assumes $\vdash f = g$
shows $\vdash \rhd f = \rhd g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence *3*: $\vdash bs(\neg f) = bs(\neg g)$ **by** (*simp add*: *BsEqvRule*)
hence *4*: $\vdash (f \wedge bs(\neg f)) = (g \wedge bs(\neg g))$ **using** *1* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*first-d-def*)
qed

lemma *FstImpFinite*:
$\vdash \rhd f \longrightarrow$ *finite*
unfolding *first-d-def* **using** *BsEqvBiMoreImpChop* **by** *fastforce*

lemma *FstWithAndImp*:
$\vdash \rhd f \wedge g \longrightarrow \rhd (f \wedge g)$
proof −
have *1*: $\vdash (\rhd f \wedge g) = ((f \wedge (bs\ (\neg f))) \wedge g)$
by (*simp add*: *first-d-def*)
have *2*: $\vdash ((f \wedge (bs\ (\neg f))) \wedge g) = (f \wedge \neg(ds\ f) \wedge g)$
using *NotDsEqvBsNot* **by** *fastforce*
have *3*: $\vdash \neg(ds\ f) \longrightarrow \neg(ds(f \wedge g))$
using *DsAndImpElimL* **by** *fastforce*
hence *4*: $\vdash f \wedge \neg(ds\ f) \wedge g \longrightarrow f \wedge g \wedge \neg(ds(f \wedge g))$
by *auto*
have *5*: $\vdash (f \wedge g \wedge \neg(ds(f \wedge g))) = ((f \wedge g) \wedge (bs\ (\neg(f \wedge g))))$
using *NotDsEqvBsNot* **by** *fastforce*
have *6*: $\vdash ((f \wedge g) \wedge (bs\ (\neg(f \wedge g)))) = \rhd(f \wedge g)$
by (*simp add*: *first-d-def*)
from *1 2 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstWithOrEqv*:
$\vdash \rhd(f \vee g) = ((\rhd f \wedge bs\ (\neg g)) \vee (\rhd g \wedge bs\ (\neg f)))$
proof −
have *1*: $\vdash \rhd(f \vee g) = ((f \vee g) \wedge bs\ (\neg(f \vee g)))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash\ (\neg(f \vee g)) =\ (\neg f \wedge \neg g)$
by *auto*
hence *3*: $\vdash bs\ (\neg(f \vee g)) =\ bs\ (\neg f \wedge \neg g)$
using *BsEqvRule* **by** *blast*
have *4*: $\vdash\ bs\ (\neg f \wedge \neg g) = (bs\ (\neg\ f) \wedge bs\ (\neg\ g))$
using *BsAndEqv* **by** *fastforce*
have *5*: $\vdash ((f \vee g) \wedge bs\ (\neg(f \vee g))) = ((f \vee g) \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g))$
using *3 4* **by** *fastforce*
have *6*: $\vdash ((f \vee g) \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g)) =$
$(((f \wedge bs\ (\neg\ f)) \wedge bs\ (\neg\ g)) \vee (g \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g)))$
by *auto*

have *7*: $\vdash ((f \wedge bs\ (\neg\ f)) \wedge bs\ (\neg\ g)) = (\triangleright f \wedge\ \ bs\ (\neg\ g))$
by (*simp add*: *first-d-def*)
have *8*: $\vdash (g \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g)) = ((g \wedge bs\ (\neg\ g)) \wedge bs\ (\neg\ f))$
by *auto*
have *9*: $\vdash ((g \wedge bs\ (\neg\ g)) \wedge bs\ (\neg\ f)) = (\triangleright g \wedge\ \ bs\ (\neg\ f))$
by (*simp add*: *first-d-def*)
have *10*: $\vdash ((f \wedge bs\ (\neg\ f)) \wedge bs\ (\neg\ g)) \vee (g \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g)) =$
$(\triangleright f \wedge\ \ bs\ (\neg\ g)) \vee (\triangleright g \wedge\ \ bs\ (\neg\ f))$
using *7 8 9* **by** *fastforce*
from *1 5 6 10* **show** *?thesis* **by** (*metis 7 8 9 int-eq*)
qed

lemma *FstFstAndEqvFstAnd*:
$\vdash \triangleright(\triangleright f \wedge g) = (\triangleright f \wedge g)$
proof −
have *1*: $\vdash (\triangleright f \wedge g) = ((f \wedge (bs\ (\neg f))) \wedge g)$ **by** (*simp add*: *first-d-def*)
hence *2*: $\vdash \triangleright f \wedge g \longrightarrow (bs\ (\neg f))$ **by** *auto*
hence *3*: $\vdash \triangleright f \wedge g \longrightarrow \triangleright f \wedge g \wedge (bs\ (\neg f))$ **by** *auto*
have *4*: $\vdash \neg\ f \longrightarrow \neg\ f \vee \neg(bs\ (\neg f)) \vee \neg\ g$ **by** *auto*
hence *5*: $\vdash bs\ (\neg\ f) \longrightarrow bs(\neg\ f \vee \neg(bs\ (\neg f)) \vee \neg\ g)$ **using** *BsImpBsRule* **by** *blast*
have *6*: $\vdash (\neg\ f \vee \neg(bs\ (\neg f)) \vee \neg\ g) = (\neg(f \wedge bs\ (\neg\ f) \wedge g))$ **by** *auto*
hence *7*: $\vdash bs(\neg\ f \vee \neg(bs\ (\neg f)) \vee \neg\ g) = bs(\neg(f \wedge bs\ (\neg\ f) \wedge g))$ **using** *BsEqvRule* **by** *blast*
have *8*: $\vdash ((f \wedge bs\ (\neg\ f)) \wedge g) = (\triangleright f \wedge g)$ **by** (*simp add*: *first-d-def*)
hence *9*: $\vdash (\neg(f \wedge bs\ (\neg\ f) \wedge g)) = (\neg(\triangleright f \wedge g))$ **by** *auto*
hence *10*: $\vdash bs\ (\neg(f \wedge bs\ (\neg\ f) \wedge g)) = bs\ (\neg(\triangleright f \wedge g))$ **using** *BsEqvRule* **by** *blast*
have *11*: $\vdash \triangleright f \wedge g \longrightarrow (\triangleright f \wedge g) \wedge\ \ bs\ (\neg(\triangleright f \wedge g))$ **using** *3 5 7 10* **by** *fastforce*
hence *12*: $\vdash \triangleright f \wedge g \longrightarrow \triangleright(\triangleright f \wedge g)$ **by** (*simp add*: *first-d-def*)
have *13*: $\vdash \triangleright(\triangleright f \wedge g) = ((\triangleright f \wedge g) \wedge\ \ bs\ (\neg(\triangleright f \wedge g)))$ **by** (*simp add*: *first-d-def*)
hence *14*: $\vdash \triangleright(\triangleright f \wedge g) \longrightarrow \triangleright f \wedge g$ **by** *auto*
from *12 14* **show** *?thesis* **by** *fastforce*
qed

lemma *FstTrue*:
$\vdash \triangleright\ \#True = empty$
proof −
have *1*: $\vdash \triangleright\ \#True = (\#True \wedge bs\ (\neg\ \#True))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash bs\ (\neg\ \#True) = (empty \vee (bi\ (\neg\ \#True)) \frown skip)$
by (*simp add*: *bs-d-def*)
have *3*: $\vdash \neg(bi\ (\neg\ \#True))$
using *BiElim* **by** *fastforce*
have *4*: $\vdash \neg((bi\ (\neg\ \#True)) \frown skip)$
by (*metis* (*no-types*, *opaque-lifting*) *BiEqvBiBi DiSkipEqvMore NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip TrueEqvTrueSChopTrue di-d-def int-simps*(*17*) *int-simps*(*19*) *int-simps*(*21*) *int-simps*(*7*) *int-simps*(*9*) *inteq-reflection itl-def*(*18*) *schop-d-def*)
have *5*: $\vdash bs\ (\neg\ \#True) = empty$
using *2 4* **by** *fastforce*
from *1 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstFalse*:
$\vdash \neg(\rhd\ \#False)$
proof −
have *1*: $\vdash \rhd\ \#False = (\#False \wedge bs\ \#True)$ **by** (*simp add*: *first-d-def*)
from *1* **show** *?thesis* **by** *auto*
qed

lemma *FstChopFalseEqvFalse*:
$\vdash \neg(\rhd f \frown \#False)$
by (*simp add*:*Valid-def itl-defs first-d-def bs-d-def*)

lemma *FstEmpty*:
$\vdash \rhd\ empty = empty$
proof −
have *1*: $\vdash \rhd\ empty = (empty \wedge bs\ (\neg\ empty))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash bs\ (\neg\ empty) = (empty \vee bi\ (\neg\ empty) \frown skip)$ **by** (*simp add*: *bs-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndEmptyEqvAndEmpty*:
$\vdash (\rhd f \wedge empty) = (f \wedge empty)$
proof −
have *1*: $\vdash (\rhd f \wedge empty) = ((f \wedge bs\ (\neg\ f)) \wedge empty)$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash bs\ (\neg\ f) = (empty \vee bi\ (\neg f) \frown skip)$ **by** (*simp add*: *bs-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstEmptyOrEqvEmpty*:
$\vdash \rhd(empty \vee f) = empty$
proof −
have *1*: $\vdash \rhd(empty \vee f) = ((\rhd empty \wedge bs\ (\neg\ f)) \vee (\rhd f \wedge bs\ (\neg\ empty)))$ **using** *FstWithOrEqv* **by** *blast*
have *2*: $\vdash\ (\neg\ empty) = more$ **by** (*simp add*: *empty-d-def*)
hence *3*: $\vdash bs\ (\neg\ empty) = bs\ more$ **using** *BsEqvRule* **by** *blast*
have *4*: $\vdash bs\ more = empty$ **using** *BsMoreEqvEmpty* **by** *blast*
have *5*: $\vdash (\rhd f \wedge bs\ (\neg\ empty)) = (\rhd f \wedge empty)$ **using** *3 4* **by** *fastforce*
have *6*: $\vdash \rhd empty = empty$ **using** *FstEmpty* **by** *blast*
hence *7*: $\vdash (\rhd empty \wedge bs\ (\neg\ f)) = (empty \wedge bs\ (\neg\ f))$ **by** *auto*
have *8*: $\vdash (empty \wedge\ bs\ (\neg\ f)) = (empty \wedge (empty \vee bi\ (\neg\ f) \frown skip))$ **by** (*simp add*:*bs-d-def*)
have *9*: $\vdash (empty \wedge (empty \vee bi\ (\neg\ f) \frown skip)) = empty$ **by** *auto*
have *10*: $\vdash (empty \wedge\ bs\ (\neg\ f)) = empty$ **using** *8 9* **by** *auto*
have *11*: $\vdash ((\rhd empty \wedge bs\ (\neg\ f)) \vee (\rhd f \wedge bs\ (\neg\ empty))) =$
$(empty \vee (\rhd f \wedge empty))$ **using** *7 10 5* **by** *fastforce*
have *12*: $\vdash (empty \vee (\rhd f \wedge empty)) = empty$ **by** *auto*
from *1 11 12* **show** *?thesis* **by** *fastforce*
qed

lemma *FstChopEmptyEqvFstChopFstEmpty*:
$\vdash (\rhd f \frown g \wedge empty) = (\rhd f \frown \rhd g \wedge empty)$
proof −
have *1*: $\vdash (\rhd f \frown g \wedge empty) = (\rhd f \wedge g \wedge empty)$ **using** *SChopEmptyAndEmpty* **by** *blast*

have *2*: $\vdash (\rhd g \wedge empty) = (g \wedge empty)$ **using** *FstAndEmptyEqvAndEmpty* **by** *blast*
hence *3*: $\vdash (\rhd f \wedge g \wedge empty) = (\rhd f \wedge \rhd g \wedge empty)$ **by** *auto*
have *4*: $\vdash (\rhd f \frown \rhd g \wedge empty) = (\rhd f \wedge \rhd g \wedge empty)$ **using** *SChopEmptyAndEmpty* **by** *blast*
from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *FstMoreEqvSkip*:
$\vdash \rhd more = skip$
proof −
have *1*: $\vdash \rhd more = (more \wedge bs\ (\neg\ more))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (more \wedge bs\ (\neg\ more)) = (more \wedge (empty \vee bi\ (\neg\ more) \frown skip))$ **by** (*simp add*:*bs-d-def*)
have *3*: $\vdash (more \wedge (empty \vee bi\ (\neg\ more) \frown skip)) = (more \wedge bi\ (\neg\ more) \frown skip)$
unfolding *empty-d-def* **by** *fastforce*
have *4*: $\vdash (more \wedge ((bi\ (\neg\ more)) \frown skip)) = ((bi\ (\neg\ more)) \frown skip)$
using *SChopSkipImpMore* **by** *fastforce*
have *5*: $\vdash ((bi\ (\neg\ more)) \frown skip) = bi\ empty \frown skip$ **by** (*simp add*: *empty-d-def*)
have *6*: $\vdash bi\ empty = empty$ **using** *BiEmptyEqvEmpty* **by** *auto*
hence *7*: $\vdash bi\ empty \frown skip = empty \frown skip$
using *LeftSChopEqvSChop* **by** *blast*
have *8*: $\vdash empty \frown skip = skip$ **using** *EmptySChop* **by** *blast*
from *1 2 3 4 5 7 8* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *FstEqvBsNotAndDi*:
$\vdash \rhd f = (bs\ (\neg\ f) \wedge di\ f)$
proof −
have *1*: $\vdash bs\ (\neg\ f) = (\neg(ds\ f))$ **by** (*simp add*: *ds-d-def*)
hence *2*: $\vdash (bs\ (\neg\ f) \wedge di\ f) = (\neg(ds\ f) \wedge di\ f)$ **by** *auto*
have *3*: $\vdash (di\ f \wedge finite) = ((ds\ f \wedge finite) \vee (f \wedge finite))$
using *OrDsEqvDi* **by** *fastforce*
have *03*: $\vdash (\neg(ds\ f)) \longrightarrow finite$
by (*metis BsEqvBiMoreImpChop Prop11 Prop12 ds-d-def int-simps*(*4*) *lift-imp-trans*)
have *4* : $\vdash (\neg(ds\ f) \wedge di\ f) = (\neg(ds\ f) \wedge ((ds\ f \wedge finite) \vee (f \wedge finite)))$
using *03 3* **by** *fastforce*
have *5* : $\vdash (\neg(ds\ f) \wedge ((ds\ f \wedge finite) \vee (f \wedge finite))) =$
$(\neg(ds\ f) \wedge f \wedge finite)$ **by** *auto*
have *6* : $\vdash (\neg(ds\ f) \wedge f \wedge finite) = (f \wedge bs\ (\neg\ f))$ **using** *1*
using *03* **by** *fastforce*
from *2 4 5 6* **show** *?thesis* **by** (*metis first-d-def int-eq*)
qed

lemma *FstOrDiEqvDi*:
$\vdash (\rhd f \vee di\ f) = di\ f$
proof −
have *1*: $\vdash (\rhd f \vee di\ f) = ((f \wedge bs\ (\neg f)) \vee di\ f)$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash ((f \wedge bs\ (\neg f)) \vee di\ f) = ((f \vee di\ f) \wedge (bs\ (\neg\ f) \vee di\ f))$ **by** *auto*
have *3*: $\vdash (f \vee di\ f) = di\ f$ **by** (*simp add*: *OrDiEqvDi*)
hence *4*: $\vdash ((f \vee di\ f) \wedge (bs\ (\neg\ f) \vee di\ f)) = (di\ f \wedge (bs\ (\neg\ f) \vee di\ f))$ **by** *auto*
have *5*: $\vdash (di\ f \wedge (bs\ (\neg\ f) \vee di\ f)) = di\ f$ **by** *auto*
from *1 2 4 5* **show** *?thesis* **by** *fastforce*

qed

lemma *FstAndDiEqvFst*:
$\vdash (\triangleright f \wedge di\ f) = \triangleright f$
proof −
have *1*: $\vdash (\triangleright f \wedge di\ f) = ((f \wedge bs\ (\neg f)) \wedge di\ f)$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (f \wedge di\ f) = f$ **by** (*meson DiIntro Prop10 Prop11*)
hence *3*: $\vdash (f \wedge bs\ (\neg f) \wedge di\ f) = (f \wedge bs\ (\neg f))$ **by** *auto*
from *1 3* **show** *?thesis* **by** (*metis first-d-def int-iffD2 int-iffI Prop12*)
qed

lemma *WPrevFiniteEqv*:
$\vdash wprev(f \wedge finite) = (\neg prev(\neg f \vee inf))$
proof −
have *1*: $\vdash (\neg\ (f \wedge \neg\ inf)) = (\neg f \vee inf)$
by *fastforce*
show *?thesis* **unfolding** *wprev-d-def finite-d-def* **using** *1*
by (*metis int-simps*(*10*) *inteq-reflection*)
qed

lemma *FPrevEqv*:
$\vdash f \frown skip = (more \wedge\ \ wprev(f \wedge finite))$
proof −
have *1*: $\vdash wprev(f \wedge finite) = (empty \vee f \frown skip)$
using *WPrevAndFiniteEqv* **by** *blast*
have *2*: $\vdash (more \wedge\ \ wprev(f \wedge finite)) = f \frown skip$
using *1* **unfolding** *empty-d-def*
by (*simp add*: *Prop01 Prop03 Prop11 Prop12 SChopSkipImpMore*)
show *?thesis*
using *2* **by** *auto*
qed

lemma *DiEqvDiFst*:
$\vdash finite \longrightarrow di\ (f) = di\ (\triangleright\ f)$
proof −
have *1*: $\vdash\ \ di\ (\triangleright\ f) = di\ (f \wedge bs\ (\neg f))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash di\ (f \wedge bs\ (\neg f)) \longrightarrow di\ f \wedge di\ (bs\ (\neg\ f))$
using *DiAndImpAnd* **by** *auto*
hence *3*: $\vdash di\ (f \wedge bs\ (\neg f)) \longrightarrow di\ f$
by *auto*
have *03*: $\vdash di\ (f \wedge bs\ (\neg f)) \longrightarrow di\ (f)$
by (*metis AndDiEqv DiImpDi Prop11 Prop12 inteq-reflection*)
have *4*: $\vdash di\ (\triangleright\ f) \longrightarrow di\ (f)$ **using** *1 03*
by *fastforce*

have *5*: $\vdash (di\ (f) \wedge empty) = (f \wedge empty)$
by (*simp add*: *DiAndEmptyEqvAndEmpty*)

have *6*: $\vdash (\rhd f \wedge empty) = (f \wedge empty)$
using *FstAndEmptyEqvAndEmpty* **by** *auto*
have *7*: $\vdash di\ (f) \wedge empty \longrightarrow \rhd f$
using *5 6* **by** *fastforce*
have *8*: $\vdash \rhd f \longrightarrow di\ (\rhd f)$
using *DiIntro* **by** *auto*
have *9*: $\vdash\ di\ (f) \wedge empty \longrightarrow di\ (\rhd f)$
using *7 8* **using** *lift-imp-trans* **by** *blast*
hence *10*: $\vdash empty \longrightarrow (di\ (f) \longrightarrow di\ (\rhd f))$
by *auto*

have *11*: $\vdash\ (\ di\ (f) \longrightarrow di\ (\rhd f))^\frown skip \longrightarrow more \wedge finite$
by (*metis FstImpFinite FstMoreEqvSkip Prop12 SChopImpFinite SChopSkipImpMore inteq-reflection*)
have *12*: $\vdash more \wedge finite \longrightarrow$
$(\ (\ di\ (f) \longrightarrow di\ (\rhd f))^\frown skip =$
$(\ (\ di\ (f))^\frown skip \longrightarrow (di\ (\rhd f))^\frown skip\ ))$
by (*simp add*: *MoreImpImpSChopSkipEqv*)
have *13*: $\vdash (more \wedge finite \wedge (\ di\ (f) \longrightarrow di\ (\rhd f))^\frown skip\ ) =$
$(more \wedge finite \wedge ((di\ (f))^\frown skip \longrightarrow (di\ (\rhd f))^\frown skip))$
using *12* **by** *fastforce*
have *14*: $\vdash\ (\ di\ (f) \longrightarrow di\ (\rhd f))^\frown skip =$
$(more \wedge finite \wedge ((di\ (f))^\frown skip \longrightarrow (di\ (\rhd f))^\frown skip))$
using *11 13* **by** *fastforce*
have *15*: $\vdash (di\ (f) \wedge finite) = ((f \wedge finite) \vee (ds\ f \wedge finite))$
using *OrDsEqvDi* **by** (*metis* (*no-types*, *opaque-lifting*) *int-eq lift-and-com*)
have *16*: $\vdash di\ (f) = (di\ (f) \wedge (bs\ (\neg f) \vee \neg(bs\ (\neg f))))$
by *auto*
have *17*: $\vdash (di\ (f) \wedge (bs\ (\neg f) \vee \neg(bs\ (\neg f)))) =$
$((di\ (f) \wedge bs\ (\neg f)) \vee (di\ (f) \wedge \neg(bs\ (\neg f))))$
by *auto*
have *171*: $\vdash bs\ (\neg f) \longrightarrow finite$
using *BsEqvBiMoreImpChop* **by** *fastforce*
have *18*: $\vdash (di\ (f) \wedge bs\ (\neg f)) = ((f \vee ds\ f) \wedge bs\ (\neg f))$
using *15 171* **by** *fastforce*
have *19*: $\vdash ((f \vee ds\ f) \wedge bs\ (\neg f)) = ((f \wedge bs\ (\neg f)) \vee (ds\ f \wedge bs\ (\neg f)))$
by *auto*
have *20*: $\vdash \neg(ds\ f \wedge bs\ (\neg f))$
by (*simp add*: *ds-d-def*)
have *21*: $\vdash ((f \wedge bs\ (\neg f)) \vee (ds\ f \wedge bs\ (\neg f))) = (f \wedge bs\ (\neg f))$
using *20* **by** *auto*
have *22*: $\vdash (di\ (f) \wedge bs\ (\neg f)) = (f \wedge bs\ (\neg f))$
using *18 19 21* **by** *fastforce*
have *23*: $\vdash (f \wedge bs\ (\neg f)) = \rhd f$
by (*simp add*: *first-d-def*)
have *24*: $\vdash (\rhd f) \longrightarrow di\ (\rhd f)$
using *DiIntro* **by** *auto*
have *25*: $\vdash (f \wedge bs\ (\neg f)) \longrightarrow di\ (\rhd f)$
using *23 24* **by** *fastforce*
have *26*: $\vdash (di\ (f) \wedge bs\ (\neg f)) \longrightarrow di\ (\rhd f)$
using *25 22* **by** *fastforce*

hence *27*: $\vdash$ $(di\ (f) \wedge bs\ (\neg\ f) \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ )) \longrightarrow di\ (\rhd f)$
by *auto*

have *28*: $\vdash$ $(di\ (f) \wedge \neg(bs\ (\neg\ f))) = (di\ (f) \wedge ds\ f)$
by (*simp add*: *ds-d-def*)
hence *29*: $\vdash$ $(di\ (f) \wedge \neg(bs\ (\neg\ f)) \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ )) =$
$(di\ (f) \wedge ds\ f \wedge finite \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ ))$
using *11* **by** *fastforce*
have *030*: $\vdash$ $finite \longrightarrow (di\ f) \frown skip = (di\ f); skip$
by (*auto simp add*: *Valid-def itl-defs*)
have *031*: $\vdash$ $finite \longrightarrow (di\ f); skip = (di\ (f \wedge finite)); skip$
by (*auto simp add*: *Valid-def itl-defs*)
have *032*: $\vdash$ $finite \longrightarrow (di\ f) \frown skip = (di(f \wedge finite)); skip$
by (*auto simp add*: *Valid-def itl-defs*)
have *30*: $\vdash$ $ds\ f = (finite \longrightarrow prev(di\ f))$
using *DsDi 030* **unfolding** *prev-d-def* **by** *fastforce*
have *033*: $\vdash$ $ds\ f = (finite \longrightarrow (di\ (f)) \frown skip)$
by (*simp add*: *DsDi*)
hence *31*: $\vdash$ $(di\ (f) \wedge ds\ f \wedge finite \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip)) =$
$(di\ (f) \wedge finite \wedge (\ (di\ (f)) \frown skip) \wedge\ (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip))$
by *auto*
have *32*: $\vdash$ $(\ di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip \longrightarrow\ ((di\ (f)) \frown skip \longrightarrow (di\ (\rhd f)) \frown skip\ )$
using *14* **by** *auto*
hence *33*: $\vdash$ $di\ (f) \wedge finite \wedge ((di\ (f)) \frown skip\ ) \wedge\ (\ di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip \longrightarrow$
$di\ (f) \wedge finite \wedge ((di\ (f)) \frown skip) \wedge ((di\ (f)) \frown skip \longrightarrow (di\ (\rhd f)) \frown skip\ )$
by *auto*
have *34*: $\vdash$ $di\ (f) \wedge finite \wedge ((di\ (f)) \frown skip) \wedge$
$((di\ (f)) \frown skip \longrightarrow (di\ (\rhd f)) \frown skip) \longrightarrow (di\ (\rhd f)) \frown skip$
by *fastforce*
have *36*: $\vdash$ $(di\ (\rhd f)) \frown skip \longrightarrow di(di\ (\rhd f))$
by (*metis AndChopA ChopImpDi lift-imp-trans schop-d-def*)
have *37*: $\vdash$ $di(di\ (\rhd f)) = di\ (\rhd f)$
using *DiEqvDiDi* **by** *fastforce*
have *38*: $\vdash$ $di\ (f) \wedge finite \wedge (di\ f) \frown skip \wedge ((di\ (f)) \frown skip \longrightarrow (di\ (\rhd f)) \frown skip) \longrightarrow di\ (\rhd f)$
using *37 36*
by (*metis 34 inteq-reflection lift-imp-trans*)
have *39*: $\vdash$ $di\ (f) \wedge finite \wedge \neg(bs\ (\neg\ f)) \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ ) \longrightarrow di\ (\rhd f)$
using *29 31 33 38 36 37* **by** *fastforce*
hence *40*: $\vdash$ $\neg(bs\ (\neg\ f)) \wedge finite \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ ) \longrightarrow (di\ (f) \longrightarrow di\ (\rhd f))$
by *fastforce*
have *41*: $\vdash$ $bs\ (\neg\ f) \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ ) \longrightarrow (di\ (f) \longrightarrow di\ (\rhd f))$
using *27* **by** *fastforce*
have *42*: $\vdash$ $(\neg(bs\ (\neg\ f)) \vee bs\ (\neg\ f)) \wedge (\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ ) \longrightarrow (di\ (f) \longrightarrow di\ (\rhd f))$
using *40 41 29* **by** *fastforce*
have *43*: $\vdash$ $(\neg(bs\ (\neg\ f)) \vee bs\ (\neg\ f))$
by *auto*
have *44*: $\vdash$ $(\ (di\ (f) \longrightarrow di\ (\rhd\ f)) \frown skip\ ) \longrightarrow (di\ (f) \longrightarrow di\ (\rhd f))$
using *42 43* **by** *fastforce*
have *45*: $\vdash$ $finite \longrightarrow di\ (f) \longrightarrow di\ (\rhd f)$
using *10 44 EmptySChopSkipInductB*[*of LIFT di* $(f) \longrightarrow di\ (\rhd\ f)$]

unfolding *prev-d-def schop-d-def* **by** *fast*
from *4 45* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteImpDiEqvImpDfEqv*:
assumes $\vdash$ *finite* $\longrightarrow$ *di f* = *di g*
shows $\vdash$ *df f* = *df g*
proof −
have *1*: $\vdash$ (*di f* $\wedge$ *finite*) = (*di g* $\wedge$ *finite*)
using *assms* **by** *fastforce*
have *2*: $\vdash$ (*di f* $\wedge$ *finite*) = *f* $\frown$ *finite*
unfolding *di-d-def*
by (*simp add*: *ChopTrueAndFiniteEqvAndFiniteChopFinite schop-d-def*)
have *3*: $\vdash$ (*di g* $\wedge$ *finite*) = *g* $\frown$ *finite*
unfolding *di-d-def*
by (*simp add*: *ChopTrueAndFiniteEqvAndFiniteChopFinite schop-d-def*)
have *4*: $\vdash$ ((*f* $\frown$ *finite*) = (*g* $\frown$ *finite*)) $\longrightarrow$ (*f*$\frown$#*True* = *g*$\frown$#*True*)
by (*metis* (*no-types*, *lifting*) *1 2 3 Prop11 SChopAssoc SChopFiniteEqvSChopTrueAndFinite TrueEqvTrueSChopTrue int-simps*(*1*) *inteq-reflection schop-d-def*)
show *?thesis* **unfolding** *df-d-def* **by** (*metis 1 2 3 4 MP inteq-reflection*)
qed

lemma *DfEqvDfFst*:
$\vdash$ *df f* = *df* ($\rhd$ *f*)
by (*simp add*: *DiEqvDiFst FiniteImpDiEqvImpDfEqv*)

lemma *FstDiEqvFst*:
$\vdash$ $\rhd$(*di f*) = $\rhd$*f*
proof −
have *1*: $\vdash$ $\rhd$(*di f*) = (*di f* $\wedge$ *bs* (¬ (*di f*))) **by** (*simp add*: *first-d-def*)
have *2*: $\vdash$ (¬ (*di f*)) = *bi* (¬ *f*) **by** (*simp add*: *NotDiEqvBiNot*)
hence *3*: $\vdash$ *bs* (¬ (*di f*)) = *bs* (*bi* (¬ *f*)) **using** *BsEqvRule* **by** *blast*
have *4*: $\vdash$ *bs* (*bi* (¬ *f*)) = *bs* (¬ *f*) **using** *BsEqvBsBi* **by** *fastforce*
hence *5*: $\vdash$ (*di f* $\wedge$ *bs* (¬ (*di f*))) = (*di f* $\wedge$ *bs* (¬ *f*)) **using** *3* **by** *fastforce*
have *05*: $\vdash$ *bs* (¬ (*di f*)) $\longrightarrow$ *finite* **using** *BsEqvBiMoreImpChop* **by** *fastforce*
have *06*: $\vdash$ *bs* (¬ *f*) $\longrightarrow$ *finite* **using** *05 3 4* **by** *fastforce*
have *6* : $\vdash$ (*di f* $\wedge$ *finite*)= ((*f* $\vee$ *ds f*) $\wedge$ *finite*) **using** *OrDsEqvDi* **by** (*metis FiniteOr inteq-reflection*)
have *07*: $\vdash$ (*di f* $\wedge$ *bs* (¬ *f*)) = (*di f* $\wedge$ *finite* $\wedge$ *bs* (¬ *f*)) **using** *06* **by** *fastforce*
have *08*: $\vdash$ (*di f* $\wedge$ *finite* $\wedge$ *bs* (¬ *f*)) = ((*f* $\vee$ *ds f*) $\wedge$ *finite* $\wedge$ *bs* (¬ *f*)) **using** *6* **by** *fastforce*
have *09*: $\vdash$ ((*f* $\vee$ *ds f*) $\wedge$ *finite* $\wedge$ *bs* (¬ *f*)) = ((*f* $\vee$ *ds f*) $\wedge$ *bs* (¬ *f*)) **using** *06* **by** *fastforce*
have *7*: $\vdash$ (*di f* $\wedge$ *bs* (¬ *f*)) = ((*f* $\vee$ *ds f*) $\wedge$ *bs* (¬ *f*)) **by** (*metis 07 08 09 inteq-reflection*)
have *8*: $\vdash$ ((*f* $\vee$ *ds f*) $\wedge$ *bs* (¬ *f*)) = ((*f* $\wedge$ *bs* (¬ *f*)) $\vee$ (*ds f* $\wedge$ *bs* (¬ *f*))) **by** *auto*
have *9*: $\vdash$ ¬(*ds f* $\wedge$ *bs* (¬ *f*)) **by** (*simp add*: *ds-d-def*)
have *10*: $\vdash$ (*f* $\wedge$ *bs* (¬ *f*)) = $\rhd$*f* **by** (*simp add*: *first-d-def*)
have *11*: $\vdash$ ((*f* $\wedge$ *bs* (¬ *f*)) $\vee$ (*ds f* $\wedge$ *bs* (¬ *f*))) = $\rhd$*f* **using** *9 10* **by** *fastforce*
from *1 5 7 8 11* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *FstDfEqvFst*:

$\vdash \rhd(df\ f) = \rhd f$
by (*metis* (*no-types*, *opaque-lifting*) *DfEqvDfFst DfEqvDiAndFinite FstDiEqvFst FstFstAndEqvFstAnd FstImpFinite Prop10 int-eq*)

lemma *DiAndFstOrEqvFstOrDiAnd*:
$\vdash (di\ f \wedge (\rhd f \vee g)) = (\rhd f \vee (di\ f \wedge g))$
proof −
have *1*: $\vdash (di\ f \wedge (\rhd f \vee g)) = (\rhd f \wedge di\ f) \vee (di\ f \wedge g)$ **by** *auto*
have *2*: $\vdash (\rhd f \wedge di\ f) = \rhd f$ **using** *FstAndDiEqvFst* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *DiOrFstAndEqvDi*:
$\vdash di\ f \vee (\rhd f \wedge g) = di\ f$
proof −
have *1*: $\vdash (di\ f \vee (\rhd f \wedge g)) = ((\rhd f \vee di\ f) \wedge (di\ f \vee g))$ **by** *auto*
have *2*: $\vdash (\rhd f \vee di\ f) = di\ f$ **using** *FstOrDiEqvDi* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *FstDiAndDiEqv*:
$\vdash \rhd(di\ f \wedge di\ g) = ((\rhd f \wedge di\ g) \vee (\rhd g \wedge di\ f))$
proof −
have *1*: $\vdash \rhd(di\ f \wedge di\ g) = ((di\ f \wedge di\ g) \wedge bs\ (\neg(di\ f \wedge di\ g)))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (\neg(di\ f \wedge di\ g)) = (bi\ (\neg f) \vee bi\ (\neg g))$ **by** (*auto simp add*: *bi-d-def*)
hence *3*: $\vdash bs\ (\neg(di\ f \wedge di\ g)) = bs(bi\ (\neg f) \vee bi\ (\neg g))$ **using** *BsEqvRule* **by** *blast*
hence *4*: $\vdash ((di\ f \wedge di\ g) \wedge bs\ (\neg(di\ f \wedge di\ g))) = (di\ f \wedge di\ g \wedge bs(bi\ (\neg f) \vee bi\ (\neg g)))$ **by** *auto*
have *5*: $\vdash (bs\ (\neg f) \vee bs\ (\neg g)) = bs(bi\ (\neg f) \vee bi\ (\neg g))$ **using** *BsOrBsEqvBsBiOrBi* **by** *blast*
hence *6*: $\vdash (di\ f \wedge di\ g \wedge bs(bi\ (\neg f) \vee bi\ (\neg g))) = (di\ f \wedge di\ g \wedge (bs\ (\neg f) \vee bs\ (\neg g)))$ **by** *auto*
have *7*: $\vdash (di\ f \wedge di\ g \wedge (bs\ (\neg f) \vee bs\ (\neg g))) = ((\ bs\ (\neg f) \wedge di\ f \wedge di\ g) \vee (di\ f \wedge bs\ (\neg g) \wedge di\ g\ ))$ **by** *auto*
have *8*: $\vdash \rhd f = (bs\ (\neg f)\ \wedge di\ f)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence *9*: $\vdash (bs\ (\neg f) \wedge di\ f \wedge di\ g) = (\rhd f \wedge di\ g)$ **by** *auto*
have *10*: $\vdash \rhd g = (bs\ (\neg g)\ \wedge di\ g)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence *11*: $\vdash (di\ f \wedge bs\ (\neg g) \wedge di\ g) = (di\ f \wedge \rhd g)$ **by** *auto*
show *?thesis*
by (*metis 11 4 6 7 9 first-d-def inteq-reflection lift-and-com*)
qed

lemma *BiNotFstEqvBiNot*:
$\vdash finite \longrightarrow bi\ (\neg\ (\rhd f)) = bi\ (\neg\ f)$
proof −
have *1*: $\vdash finite \longrightarrow di\ f = di\ (\rhd f)$ **using** *DiEqvDiFst* **by** *blast*
hence *2*: $\vdash finite \longrightarrow (\neg(di\ f)) = (\neg(\ di\ (\rhd f)))$ **by** *auto*
from *1 2* **show** *?thesis* **using** *NotDiEqvBiNot* **by** *fastforce*
qed

lemma *BsNotFstEqvBsNot*:

$\vdash$ *bs* (¬ (▷*f*)) = *bs* (¬ *f*)
proof −
have *1*: $\vdash$ *bs* (¬ (▷*f*)) = (*empty* ∨ *bi* (¬ (▷*f*))⌢*skip*) **by** (*simp add*: *bs-d-def*)
have *2*: $\vdash$ *finite*⟶*bi* (¬ (▷*f*)) = *bi* (¬ *f*) **using** *BiNotFstEqvBiNot* **by** *fastforce*
hence *3*: $\vdash$ *bi* (¬ (▷*f*))⌢*skip* = *bi* (¬ *f*)⌢*skip* **using** *LeftSChopEqvSChop*
by (*simp add*: *FiniteImpAnd LeftChopEqvChop schop-d-def*)
hence *4*: $\vdash$ (*empty* ∨ *bi* (¬ (▷*f*))⌢*skip*) = (*empty* ∨ *bi* (¬ *f*)⌢*skip*) **by** *auto*
from *1 4* **show** *?thesis* **by** (*metis BsEqvEmptyOrBiSChopSkip inteq-reflection*)
qed

lemma *FstState*:
$\vdash$ ▷ (*init w*) = (*empty* ∧ *init w*)
proof −
have *1*: $\vdash$ ▷ (*init w*) = (*init w* ∧ *bs* (¬(*init w*))) **by** (*simp add*: *first-d-def*)
hence *2*: $\vdash$ ▷ (*init w*) ⟶ *init w* **by** *auto*
have *3*: $\vdash$ *init w* ∧ *finite* ⟶ *bs* (*init w*) **using** *StateImpBs* **by** *auto*
have *4*: $\vdash$ ▷ (*init w*) ⟶ *bs* (*init w*) **using** *2 3*
using *FstImpFinite* **by** *fastforce*
have *5*: $\vdash$ ▷ (*init w*) ⟶ *bs* (¬(*init w*)) **using** *1* **by** *auto*
have *6*: $\vdash$ ▷ (*init w*) ⟶ *bs* (*init w*) ∧ *bs* (¬(*init w*)) **using** *4 5* **by** *fastforce*
have *7*: $\vdash$ (*bs* (*init w*) ∧ *bs* (¬(*init w*))) = (*bs*((*init w*) ∧ ¬(*init w*))) **using** *BsAndEqv* **by** *blast*
have *8*: $\vdash$ ((*init w*) ∧ ¬(*init w*)) = *#False* **by** *auto*
hence *9*: $\vdash$ (*bs*((*init w*) ∧ ¬(*init w*))) = *bs #False* **using** *BsEqvRule* **by** *blast*
have *10*: $\vdash$ *bs #False* = *empty* **using** *BsFalseEqvEmpty* **by** *auto*
have *11*: $\vdash$ ▷ (*init w*) ⟶ *empty* **using** *10 9 7 6* **by** *fastforce*
have *12*: $\vdash$ ▷ (*init w*) ⟶ *empty* ∧ *init w* **using** *11 2* **by** *fastforce*
have *13*: $\vdash$ *empty* ∧ *init w* ⟶ *empty* **by** *auto*
hence *14*: $\vdash$ *empty* ∧ *init w* ⟶ *empty* ∨ *bi* (¬(*init w*))⌢*skip* **by** *auto*
hence *15*: $\vdash$ *empty* ∧ *init w* ⟶ *bs* (¬(*init w*)) **by** (*simp add*: *bs-d-def*)
have *16*: $\vdash$ *empty* ∧ *init w* ⟶ *init w* **by** *auto*
have *17*: $\vdash$ *empty* ∧ *init w* ⟶ *init w* ∧ *bs* (¬(*init w*)) **using** *16 15* **by** *auto*
hence *18*: $\vdash$ *empty* ∧ *init w* ⟶ ▷(*init w*) **by** (*simp add*: *first-d-def*)
from *12 18* **show** *?thesis* **by** *fastforce*
qed

lemma *FstStateAndBsNotEmpty*:
$\vdash$ (▷ (*init w*) ∧ *bs* (¬ *empty*)) = ▷ (*init w*)
proof −
have *1*: $\vdash$ (▷ (*init w*) ∧ *bs* (¬ *empty*)) = (▷ (*init w*) ∧ *bs more*)
by (*simp add*: *empty-d-def*)
have *2*: $\vdash$ (▷ (*init w*) ∧ *bs more*) = (▷ (*init w*) ∧ *empty*)
using *BsMoreEqvEmpty* **by** *fastforce*
have *3*: $\vdash$ ▷ (*init w*) = (*empty* ∧ (*init w*))
using *FstState* **by** *blast*
hence *4*: $\vdash$ (▷ (*init w*) ∧ *empty*) = (*empty* ∧ (*init w*) ∧ *empty*)
by *auto*
have *5*: $\vdash$ (*empty* ∧ (*init w*) ∧ *empty*) = (*empty* ∧ (*init w*))
by *auto*
have *6*: $\vdash$ (*empty* ∧ (*init w*)) = ▷(*init w*)
using *FstState* **by** *fastforce*

from *1 2 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstStateImpFstStateOr*:
$\vdash \triangleright(init\ w) \longrightarrow \triangleright(init\ w \vee f)$
proof −
have *1*: $\vdash \triangleright(init\ w) = (empty \wedge init\ w)$
using *FstState* **by** *blast*
have *2*: $\vdash\ (empty \wedge init\ w) =\ (empty \wedge (empty \vee bi\ (\neg f)^\frown skip) \wedge init\ w)$
by *auto*
have *3*: $\vdash\ (empty \wedge (empty \vee bi\ (\neg f)^\frown skip) \wedge init\ w) =$
$(empty \wedge bs\ (\neg f) \wedge init\ w)$
by (*simp add*: *bs-d-def*)
have *4*: $\vdash\ (empty \wedge bs\ (\neg f) \wedge init\ w) = (empty \wedge init\ w \wedge bs\ (\neg f))$
by *auto*
have *5*: $\vdash (empty \wedge init\ w)\ = \triangleright\ (init\ w)$
using *FstState* **by** *fastforce*
hence *6*: $\vdash (empty \wedge init\ w \wedge bs\ (\neg f)) = (\triangleright\ (init\ w) \wedge bs\ (\neg f))$
by *auto*
have *7*: $\vdash \triangleright\ (init\ w) \wedge bs\ (\neg f) \longrightarrow (\triangleright\ (init\ w) \wedge bs\ (\neg f)) \vee (\triangleright f \wedge bs\ (\neg(init\ w)))$
by *auto*
have *8*: $\vdash \triangleright(init\ w \vee f) = ((\triangleright\ (init\ w) \wedge bs\ (\neg f)) \vee (\triangleright f \wedge bs\ (\neg(init\ w))))$
using *FstWithOrEqv* **by** *blast*
show *?thesis* **using** *1 3 8* **by** *fastforce*
qed

lemma *FstLenSame*:
$(\forall\ \sigma.\ (\sigma \models\ di\ (\triangleright f \wedge len(i)) \wedge di\ (\triangleright f \wedge len(j))) \longrightarrow (i{=}j))$
by (*simp add*: *DiLenFstsem FstLenSamesem*)

lemma *FstLenSame-1*:
$\vdash\ di\ (\triangleright f \wedge len(i)) \wedge di\ (\triangleright f \wedge len(j)) \longrightarrow (\#i{=}\#j)$
using *FstLenSame Valid-def* **by** *fastforce*

lemma *FstAndLenSame*:
$(\forall\ \sigma.\ (\sigma \models\ di\ ((\triangleright f \wedge g1) \wedge len(i)) \wedge di\ ((\triangleright f \wedge g2) \wedge len(j))) \longrightarrow (i{=}j))$
using *linorder-neqE-nat* **by** (*simp add*: *DiLenFstAndsem*) *blast*

lemma *FstAndLenSame-1*:
$\vdash\ di\ ((\triangleright f \wedge g1) \wedge len(i)) \wedge di\ ((\triangleright f \wedge g2) \wedge len(j)) \longrightarrow (\#i{=}\#j)$
using *FstAndLenSame Valid-def* **by** *fastforce*

lemma *FstLenSameChop*:
$(\forall\ \sigma.\ (\sigma \models\ ((\triangleright f \wedge g1) \wedge len(i));h1 \wedge\ ((\triangleright f \wedge g2) \wedge len(j));h2) \longrightarrow (i{=}j))$
proof
fix σ
show $(\sigma \models\ ((\triangleright f\ \wedge g1) \wedge len(i));h1 \wedge\ ((\triangleright f \wedge g2) \wedge len(j));h2) \longrightarrow (i{=}j)$
proof
assume *0*: $(\sigma \models\ ((\triangleright f \wedge g1) \wedge len(i));h1 \wedge\ ((\triangleright f \wedge g2) \wedge len(j));h2)$

have *1*: $(\sigma \models ((\rhd f \wedge g1) \wedge len(i));h1)$ **using** *0* **by** *auto*
have *2*: $(\sigma \models ((\rhd f \wedge g1) \wedge len(i));h1) \longrightarrow$
$(\sigma \models ((\rhd f \wedge g1) \wedge len(i));\#True)$ **by** (*metis ChopImpDi Valid-def di-d-def unl-lift2*)
have *3*: $(\sigma \models di((\rhd f \wedge g1) \wedge len(i)))$ **using** *1 2* **by** (*simp add*: *di-d-def*)
have *4*: $(\sigma \models ((\rhd f \wedge g2) \wedge len(j));h2)$ **using** *0* **by** *auto*
have *5*: $(\sigma \models ((\rhd f \wedge g2) \wedge len(j));h2) \longrightarrow$
$(\sigma \models ((\rhd f \wedge g2) \wedge len(j));\#True)$ **by** (*metis ChopImpDi Valid-def di-d-def unl-lift2*)
have *6*: $(\sigma \models di((\rhd f \wedge g2) \wedge len(j)))$ **using** *4 5* **by** (*simp add*: *di-d-def*)
have *7*: $(\sigma \models di((\rhd f \wedge g1) \wedge len(i)) \wedge di((\rhd f \wedge g2) \wedge len(j)))$ **using** *3 6* **by** *auto*
thus $(i{=}j)$ **using** *FstAndLenSame* **by** *blast*
qed
qed

lemma *FstLenSameChop-1*:
$\vdash ((\rhd f \wedge g1) \wedge len(i));h1 \wedge ((\rhd f \wedge g2) \wedge len(j));h2 \longrightarrow (\#i{=}\#j)$
using *FstLenSameChop Valid-def* **by** *fastforce*

lemma *DiImpExistsOneDiLenAndFst*:
assumes *nfinite* σ
$(\sigma \models di\ f)$
shows $(\exists!\ k.\ (\sigma \models di(\ \rhd f \wedge len(k))))$
proof −
have *1*: $(\sigma \models di\ (\rhd f))$
using *assms DiEqvDiFst*
by (*metis* (*no-types*, *lifting*) *FiniteImpAnd finite-defs inteq-reflection unl-lift2*)
have *2*: $(\sigma \models \rhd f) = (\ (\sigma \models \rhd f) \wedge (\exists k.\ (\sigma \models len(k))))$
using *AndExistsLen*[*of TEMP* $\rhd f$]
using *assms len-defs nfinite-conv-nlength-enat* **by** *blast*
have *3*: $((\sigma \models \rhd f) \wedge (\exists k.\ (\sigma \models len(k)))) =$
$(\exists k.\ (\sigma \models \rhd f) \wedge (\sigma \models len(k)))$
by *auto*
have *4*: $(\sigma \models di(\rhd f)) = (\exists k.\ (\sigma \models di(\rhd f \wedge len(k))))$
using *assms* **by** (*metis DiLensem itl-defs*(*10*))
have *5*: $(\exists k.\ (\sigma \models di(\rhd f \wedge len(k))))$
using *1* **using** *4* **by** *auto*
then obtain *i* **where** *6*: $(\sigma \models di(\rhd f \wedge len(i)))$ **by** *blast*
from *5* **obtain** *j* **where** *7*: $(\sigma \models di(\rhd f \wedge len(j)))$ **by** *blast*
have *8*: $(\sigma \models di(\rhd f \wedge len(i))) \wedge (\sigma \models di(\rhd f \wedge len(j)))$
using *6 7* **by** *auto*
hence *9*: $(\sigma \models di(\rhd f \wedge len(i)) \wedge di(\rhd f \wedge len(j)))$
by *simp*
hence *10*: $i{=}j$
using *FstLenSame* **by** *blast*
have *11*: $\bigwedge j.\ (\sigma \models di(\rhd f \wedge len(j))) \longrightarrow (j{=}i)$
using *9 10* **using** *FstLenSame* **by** *auto*
thus $(\exists!k.\ (\sigma \models di\ (\rhd f \wedge len\ k))\ )$
using *11 5* **by** *blast*
qed

lemma *DiImpExistsOneDiLenAndFst-1*:
$\vdash$ *finite* $\longrightarrow$ *di f* $\longrightarrow$ ($\exists!$ *k*. (*di*($\rhd f \wedge len(k)$)))
using *DiImpExistsOneDiLenAndFst* **unfolding** *Valid-def finite-defs* **by** *fastforce*

lemma *LFstAndDist-help*:
($\sigma \models((\rhd f \wedge g1) \wedge len(k));h1 \ \wedge ((\rhd f \wedge g2) \wedge len(k));h2) =$
($\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge len(k));(h1 \wedge h2)$)
using *LFixedAndDistr* **by** *fastforce*

lemma *LFstAndDist-help-1*:
($\exists$ *k*. ($\sigma \models((\rhd f \wedge g1) \wedge len(k));h1 \ \wedge ((\rhd f \wedge g2) \wedge len(k));h2)) =$
($\exists$ *k*. ($\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge len(k));(h1 \wedge h2)$))
proof
assume *0*: $\exists k.\ \sigma \models \ ((\rhd f \wedge g1) \wedge len\ k)\ ;\ h1 \ \wedge \ ((\rhd f \wedge g2) \wedge len\ k)\ ;\ h2$
obtain *k* **where** *1*: $\sigma \models \ ((\rhd f \wedge g1) \wedge len\ k)\ ;\ h1 \ \wedge \ ((\rhd f \wedge g2) \wedge len\ k)\ ;\ h2$
using *0* **by** *auto*
hence *2*: ($\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge len(k));(h1 \wedge h2)$)
using *LFstAndDist-help* **by** *blast*
show ($\exists$ *k*. ($\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge len(k));(h1 \wedge h2)$))
using *2* **by** *auto*
next
assume *3*: ($\exists$ *k*. ($\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge len(k));(h1 \wedge h2)$))
obtain *k* **where** *4*: ($\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge len(k));(h1 \wedge h2)$)
using *3* **by** *auto*
hence *5*: ($\sigma \models((\rhd f \wedge g1) \wedge len(k));h1 \ \wedge ((\rhd f \wedge g2) \wedge len(k));h2$)
using *LFstAndDist-help* **by** *blast*
show ($\exists$ *k*. ($\sigma \models((\rhd f \wedge g1) \wedge len(k));h1 \ \wedge ((\rhd f \wedge g2) \wedge len(k));h2$))
using *5* **by** *auto*
qed

lemma *LFstAndDistrsem*:
(($\sigma \models((\rhd f \wedge g1);h1 \wedge (\rhd f \wedge g2);h2) = (\rhd f \wedge g1 \wedge g2);(h1 \wedge h2)$))
proof −
have *01*: ($\sigma \models(\rhd f \wedge g1);h1) = (\sigma \models(\rhd f \wedge g1)^\frown h1$)
using *FstImpFinite schop-d-def*
by (*metis* (*no-types*, *lifting*) *FstFstAndEqvFstAnd Prop10 inteq-reflection*)
have *02*: ($\sigma \models(\rhd f \wedge g2);h2) = (\sigma \models(\rhd f \wedge g2)^\frown h2$)
using *FstImpFinite schop-d-def*
by (*metis* (*no-types*, *lifting*) *FstFstAndEqvFstAnd Prop10 inteq-reflection*)
have *1*: ($\sigma \models(\rhd f \wedge g1)^\frown h1) = (\exists\ i.\ (\sigma \models((\rhd f \wedge g1) \wedge len(i));h1)$)
using *AndExistsLenChop*[*of TEMP* ($\rhd f \wedge g1$)] **by** *fastforce*
have *2*: ($\sigma \models(\rhd f \wedge g2)^\frown h2) = (\exists\ j.\ (\sigma \models((\rhd f \wedge g2) \wedge len(j));h2)$)
using *AndExistsLenChop*[*of TEMP* ($\rhd f \wedge g2$)] **by** *fastforce*
have *3*: ($\sigma \models(\rhd f \wedge g1)^\frown h1 \wedge (\rhd f \wedge g2)^\frown h2) =$
(($\exists\ i\ j.\ (\sigma \models((\rhd f \wedge g1) \wedge len(i));h1 \ \wedge ((\rhd f \wedge g2) \wedge len(j));h2)$)
)
using *1 2* **by** *auto*
have *4*: (($\exists\ i\ j.\ (\sigma \models((\rhd f \wedge g1) \wedge len(i));h1 \ \wedge ((\rhd f \wedge g2) \wedge len(j));h2)$)
) =
(($\exists\ k.\ (\sigma \models((\rhd f \wedge g1) \wedge len(k));h1 \ \wedge ((\rhd f \wedge g2) \wedge len(k));h2)$)

```
        )
  using FstLenSameChop by blast
  have 5: (∃ k. (σ ⊨((▷f ∧ g1) ∧ len(k));h1  ∧ ((▷f ∧ g2) ∧ len(k));h2)) =
          (∃ k. (σ ⊨(((▷f ∧ g1) ∧ (▷f ∧ g2)) ∧ len(k));(h1 ∧ h2) ))
  using LFstAndDist-help-1 by blast
  have 6 :  (∃ k. (σ ⊨(((▷f ∧ g1) ∧ (▷f ∧ g2)) ∧ len(k));(h1 ∧ h2) )) =
           (σ ⊨((▷f ∧ g1) ∧ (▷f ∧ g2))⌢(h1 ∧ h2))
  using AndExistsLenChop[of TEMP ((▷ f ∧ g1) ∧ ▷ f ∧ g2)] by fastforce
  have 7 : (σ ⊨((▷f ∧ g1) ∧ (▷f ∧ g2));(h1 ∧ h2)) =
          (σ ⊨(▷f ∧ g1 ∧ g2);(h1 ∧ h2))
  by (auto simp add: chop-defs)
  have 8:  (σ ⊨(▷f ∧ g1 ∧ g2)⌢(h1 ∧ h2)) = (σ ⊨(▷f ∧ g1 ∧ g2);(h1 ∧ h2))
    using FstImpFinite schop-d-def
    by (metis (no-types, lifting) FstFstAndEqvFstAnd Prop10 inteq-reflection)
  have 9: (σ ⊨(▷f ∧ g1)⌢h1 ∧ (▷f ∧ g2)⌢h2) =
          (σ ⊨(▷f ∧ g1 ∧ g2);(h1 ∧ h2))
      using 01 02 3 4 5 6 7 8
      by (metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite Prop10 inteq-reflection schop-d-def)
  show ?thesis
  unfolding schop-d-def using 01 02 9 by auto
qed


lemma LFstAndDistr:
⊢ ((▷f ∧ g1);h1 ∧ (▷f ∧ g2);h2) = (▷f ∧ g1 ∧ g2);(h1 ∧ h2)
using  LFstAndDistrsem by fastforce

lemma LFstAndDistrA:
⊢ ((▷f ∧ g1);h ∧ (▷f ∧ g2);h) = (▷f ∧ g1 ∧ g2);h
proof −
 have 1: ⊢ ((▷f ∧ g1);h ∧ (▷f ∧ g2);h) = (▷f ∧ g1 ∧ g2);(h ∧ h)  using LFstAndDistr by blast
 have 2: ⊢ (▷f ∧ g1 ∧ g2);(h ∧ h) = (▷f ∧ g1 ∧ g2);h  by auto
 from 1 2 show ?thesis by auto
qed

lemma LFstAndDistrB:
⊢ ((▷f ∧ g);h1 ∧ (▷f ∧ g);h2) = (▷f ∧ g);(h1 ∧ h2)
proof −
 have 1: ⊢ ((▷f ∧ g);h1 ∧ (▷f ∧ g);h2) = (▷f ∧ g ∧ g);(h1 ∧ h2)  using LFstAndDistr by blast
 have 2: ⊢ (▷f ∧ g ∧ g);(h1 ∧ h2) = (▷f ∧ g);(h1 ∧ h2)  by auto
 from 1 2 show ?thesis by auto
qed

lemma LFstAndDistrC:
  ⊢ ((▷f );h1 ∧ (▷f );h2) = (▷f);(h1 ∧ h2)
proof −
 have 1: ⊢ ((▷f ∧ #True);h1 ∧ (▷f ∧ #True);h2) = (▷f ∧ #True ∧ #True);(h1 ∧ h2)
     using LFstAndDistr by blast
 have 2: ⊢ (▷f ∧ #True);h1 = (▷f );h1
     by auto
```

have *3*: $\vdash (\triangleright f \wedge \# True);h2 = (\triangleright f\ );h2$
by *auto*
have *4*: $\vdash (\triangleright f \wedge \# True \wedge \# True);(h1 \wedge h2) = (\triangleright f);(h1 \wedge h2)$
by *auto*
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *LFstAndDistrD*:
$\vdash (di(\triangleright f \wedge g1) \wedge di(\triangleright f \wedge g2)) = di(\triangleright f \wedge g1 \wedge g2)$
proof −
have *1*: $\vdash ((\triangleright f \wedge g1);\# True \wedge (\triangleright f \wedge g2);\# True) = (\triangleright f \wedge g1 \wedge g2);(\# True \wedge \# True)$
using *LFstAndDistr* **by** *blast*
have *2*: $\vdash (\triangleright f \wedge g1);\# True = di(\triangleright f \wedge g1)$
by (*simp add*: *di-d-def*)
have *3*: $\vdash (\triangleright f \wedge g2);\# True = di(\triangleright f \wedge g2)$
by (*simp add*: *di-d-def*)
have *4*: $\vdash (\triangleright f \wedge g1 \wedge g2);(\# True \wedge \# True) = di(\triangleright f \wedge g1 \wedge g2)$
by (*simp add*: *di-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotFstChop*:
$\vdash (\neg(\triangleright f\ ;g)) = (\neg(di\ (\triangleright f)) \vee (\triangleright f;(\neg g)))$
proof −
have *1*: $\vdash g \longrightarrow \# True$ **by** *auto*
hence *2*: $\vdash \triangleright f;g \longrightarrow \triangleright f;\# True$ **using** *RightChopImpChop* **by** *blast*
hence *3*: $\vdash \triangleright f;g \longrightarrow di(\triangleright f)$ **by** (*simp add*:*di-d-def*)
hence *4*: $\vdash \neg(di(\triangleright f)) \longrightarrow \neg(\triangleright f;g)$ **by** *auto*
have *5*: $\vdash (\triangleright f;\ (\neg g) \longrightarrow \neg(\triangleright f;g)) = ((\triangleright f;\ (\neg g)) \wedge (\triangleright f;g) \longrightarrow \# False\ )$ **by** *auto*
have *6*: $\vdash ((\triangleright f;\ (\neg g)) \wedge (\triangleright f;g)) = \triangleright f;(\neg g \wedge g)$ **using** *LFstAndDistrC* **by** *blast*
have *06*: $\vdash \triangleright f \longrightarrow finite$ **by** (*simp add*: *FstImpFinite*)
have *7*: $\vdash \neg(\triangleright f;(\neg g \wedge g))$ **using** *06* **by** (*auto simp add*: *Valid-def itl-defs*)
have *8*: $\vdash \triangleright f;\ (\neg g) \longrightarrow \neg(\triangleright f;g)$ **using** *5 6 7* **by** *fastforce*
have *9*: $\vdash \neg(di(\triangleright f)) \vee (\triangleright f;\ (\neg g)) \longrightarrow \neg(\triangleright f;g)$ **using** *4 8* **by** *fastforce*
have *10*: $\vdash di(\triangleright f) \vee \neg(di(\triangleright f))$ **by** *auto*
hence *11*: $\vdash (\triangleright f;\# True) \vee \neg(di(\triangleright f))$ **by** (*simp add*: *di-d-def*)
hence *12*: $\vdash (\triangleright f;(g \vee \neg g)) \vee \neg(di(\triangleright f))$ **by** *auto*
have *13*: $\vdash (\triangleright f;(g \vee \neg g)) = ((\triangleright f;g) \vee (\triangleright f;(\neg g)))$ **using** *ChopOrEqv* **by** *fastforce*
have *14*: $\vdash ((\triangleright f;g) \vee (\triangleright f;(\neg g))) \vee \neg(di(\triangleright f))$ **using** *12 13* **by** *fastforce*
hence *15*: $\vdash \neg(\triangleright f;g) \longrightarrow \neg(di(\triangleright f)) \vee (\triangleright f;\ (\neg g))$ **by** *auto*
from *9 15* **show** *?thesis* **by** *fastforce*
qed

lemma *BsNotFstChop*:
$\vdash bs(\neg(\triangleright f;g)) = (empty \vee (finite \wedge \neg(di\ (\triangleright f))) \vee\ \triangleright f;(bs(\neg g)))$
proof −
have *01*: $\vdash (\triangleright f;g) = (\triangleright f \frown g)$
by (*simp add*: *FstImpFinite LeftChopEqvChop Prop10 schop-d-def*)
have *1*: $\vdash bs(\neg(\triangleright f;g))= (empty \vee bi(\neg(\triangleright f \frown g)) \frown skip)$

using *01* **unfolding** *bs-d-def* **by** (*metis WPrevAndFiniteEqv inteq-reflection*)
have *000*: $\vdash$ (*empty* $\vee$ *bi*($\neg$($\triangleright$*f*$\frown$*g*))$\frown$*skip*) = (*empty* $\vee$ *bf*($\neg$($\triangleright$*f*$\frown$*g*))$\frown$*skip*)
by (*metis FiniteImpAnd FiniteImpBfEqvBi WPrevAndFiniteEqv inteq-reflection*)
have *2*: $\vdash$ (*empty* $\vee$ *bf*($\neg$($\triangleright$*f*$\frown$*g*))$\frown$*skip*) = (*empty* $\vee$ ($\neg$(*df*($\triangleright$*f*$\frown$*g*)))$\frown$*skip*)
by (*simp add*: *bf-d-def*)
have *3*: $\vdash$ (*empty* $\vee$ ($\neg$(*df*($\triangleright$*f*$\frown$*g*)))$\frown$*skip*) = (*empty* $\vee$ ($\neg$(($\triangleright$*f*$\frown$*g*)$\frown$#*True*))$\frown$*skip*)
by (*simp add*: *df-d-def*)
have *4*: $\vdash$ ($\neg$(($\triangleright$*f*$\frown$*g*)$\frown$#*True*))$\frown$*skip* = ($\neg$($\triangleright$*f*$\frown$(*g*$\frown$#*True*)))$\frown$*skip*
by (*metis LeftSChopEqvSChop SChopAssoc int-simps*(*14*) *inteq-reflection*)
hence *5*: $\vdash$ (*empty* $\vee$ ($\neg$(($\triangleright$*f*$\frown$*g*)$\frown$#*True*))$\frown$*skip*) = (*empty* $\vee$ ($\neg$($\triangleright$*f*$\frown$(*g*$\frown$#*True*)))$\frown$*skip*)
by *auto*
have *6*: $\vdash$ (*empty* $\vee$ ($\neg$($\triangleright$*f*$\frown$(*g*$\frown$#*True*)))$\frown$*skip*) = (*empty* $\vee$ ($\neg$($\triangleright$*f*$\frown$*df*(*g*)))$\frown$*skip*)
by (*simp add*: *df-d-def*)
have *7*: $\vdash$ (*empty* $\vee$ ($\neg$($\triangleright$*f*$\frown$*df*(*g*)))$\frown$*skip*) = (*empty* $\vee$ $\neg$($\neg$(($\neg$($\triangleright$*f*$\frown$*df*(*g*)))$\frown$*skip*)))
by *auto*
have *8*: $\vdash$ ($\neg$($\neg$(($\neg$($\triangleright$*f*$\frown$*df*(*g*)))$\frown$*skip*))) = ($\neg$(*empty* $\vee$ *inf* $\vee$ ($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*))
using *NotNotSChopSkip* **by** *fastforce*
hence *9*: $\vdash$ (*empty* $\vee$ $\neg$($\neg$(($\neg$($\triangleright$*f*$\frown$*df*(*g*)))$\frown$*skip*))) = (*empty* $\vee$ $\neg$(*empty* $\vee$ *inf* $\vee$ ($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*))
by *auto*
have *09*: $\vdash$ ($\neg$(*empty* $\vee$ *inf* $\vee$ ($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*)) = (*more* $\wedge$ *finite* $\wedge$ $\neg$(($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*))
unfolding *empty-d-def finite-d-def* **by** *fastforce*
have *10*: $\vdash$ (*empty* $\vee$ $\neg$(*empty* $\vee$ *inf* $\vee$ ($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*)) = (*empty* $\vee$ (*more* $\wedge$ *finite* $\wedge$ $\neg$(($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*)))
using *6 7 8 9 09* **by** *auto*
have *11*: $\vdash$ (*empty* $\vee$ (*more* $\wedge$ *finite* $\wedge$ $\neg$(($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*))) = (*empty* $\vee$ (*finite* $\wedge$ $\neg$(($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*)))
by (*auto simp add*: *empty-d-def*)
have *12*: $\vdash$ (*empty* $\vee$ (*finite* $\wedge$ $\neg$(($\triangleright$*f*$\frown$*df*(*g*))$\frown$*skip*))) = (*empty* $\vee$ (*finite* $\wedge$ $\neg$($\triangleright$*f*$\frown$(*df*(*g*)$\frown$*skip*))))
by (*metis 11 SChopAssoc inteq-reflection*)
have *012*: $\vdash$ ($\neg$($\triangleright$*f*$\frown$(*inf* $\vee$ *df*(*g*)$\frown$*skip*))) = ($\neg$($\triangleright$*f*$\frown$*inf*) $\wedge$ $\neg$($\triangleright$*f*$\frown$(*df*(*g*)$\frown$*skip*)))
using *SChopOrEqv* **by** *fastforce*
have *013*: $\vdash$ ($\neg$($\triangleright$*f*$\frown$*inf*)) = ($\neg$ *di* ($\triangleright$ *f*) $\vee$ $\triangleright$ *f*$\frown$($\neg$ *inf*))
by (*metis FstImpFinite NotFstChop Prop10 inteq-reflection schop-d-def*)
have *014*:$\vdash$ (*empty* $\vee$ (*finite* $\wedge$ $\neg$($\triangleright$*f*$\frown$(*df*(*g*)$\frown$*skip*)))) =
(*empty* $\vee$ (*finite* $\wedge$ ($\neg$(*di* ($\triangleright$*f*)) $\vee$ ($\triangleright$*f*$\frown$($\neg$(*df*(*g*)$\frown$*skip*))))))
using *NotFstChop*
by (*metis FstImpFinite Prop10 int-simps*(*1*) *inteq-reflection schop-d-def*)
have *015*: $\vdash$ ($\neg$(*df*(*g*)$\frown$*skip*)) = (*empty* $\vee$ *inf* $\vee$ *bf*($\neg$*g*)$\frown$*skip*)
by (*metis NotDfEqvBfNot NotSChopNotSkip inteq-reflection*)
have *16*: $\vdash$ ($\triangleright$*f*$\frown$($\neg$(*df*(*g*)$\frown$*skip*))) = ($\triangleright$*f*$\frown$(*empty* $\vee$ *inf* $\vee$ *bf*($\neg$*g*)$\frown$*skip*))
by (*simp add*: *015 RightSChopEqvSChop*)
have *0160*: $\vdash$ *bf*($\neg$*g*)$\frown$*skip* = *bi*($\neg$*g*)$\frown$*skip*
by (*simp add*: *FiniteImpAnd FiniteImpBfEqvBi LeftChopEqvChop schop-d-def*)
have *016*: $\vdash$ (*empty* $\vee$ *inf* $\vee$ *bf*($\neg$*g*)$\frown$*skip*) = (*inf* $\vee$ *bs* ($\neg$*g*))
unfolding *bs-d-def* **using** *0160* **by** *fastforce*
have *017*: $\vdash$ ($\triangleright$*f*$\frown$($\neg$(*df*(*g*)$\frown$*skip*))) = ($\triangleright$*f*$\frown$ (*inf* $\vee$ *bs* ($\neg$*g*)))
by (*metis 016 16 inteq-reflection*)
have *018*:$\vdash$ ($\triangleright$*f*$\frown$ (*inf* $\vee$ *bs* ($\neg$*g*))) = ($\triangleright$*f*$\frown$*inf* $\vee$ $\triangleright$*f*$\frown$(*bs*($\neg$*g*)))
by (*simp add*: *SChopOrEqv*)
have *019*: $\vdash$ (*finite* $\wedge$ ($\neg$(*di* ($\triangleright$*f*)) $\vee$ ($\triangleright$*f*$\frown$($\neg$(*df*(*g*)$\frown$*skip*))))) =

$((finite \wedge \neg(di\ (\rhd f))) \vee (finite \wedge (\rhd f \frown (\neg(df(g) \frown skip)))))$
by *force*
have *020*: $\vdash (finite \wedge (\rhd f \frown (\neg(df(g) \frown skip)))) = (finite \wedge (\rhd f \frown inf \vee \rhd f \frown (bs(\neg g))))$
using *017 018* **by** *fastforce*
have *021*:$\vdash \rhd f \frown inf \longrightarrow inf$
by (*metis AndChopB FiniteChopInfEqvInf Prop11 lift-imp-trans schop-d-def*)
have *022*: $\vdash (finite \wedge (\rhd f \frown inf \vee \rhd f \frown (bs(\neg g)))) = (finite \wedge \rhd f \frown (bs(\neg g)))$
using *021* **unfolding** *finite-d-def* **by** *fastforce*
have *023*: $\vdash \rhd f \frown (bs(\neg g)) \longrightarrow finite$
by (*metis AndChopB EmptyImpFinite FiniteChopSkipImpFinite Prop02 SChopImpFinite bs-d-def lift-imp-trans schop-d-def*)
have *024*: $\vdash (finite \wedge \rhd f \frown (bs(\neg g))) = \rhd f \frown (bs(\neg g))$
using *023* **by** *fastforce*
have *025*: $\vdash (empty \vee (finite \wedge (\neg(di\ (\rhd f)) \vee (\rhd f \frown (\neg(df(g) \frown skip)))))) =$
$(empty \vee (finite \wedge \neg(di\ (\rhd f))) \vee \rhd f \frown (bs(\neg g)))$
by (*metis 014 019 020 022 024 inteq-reflection*)
have *026*: $\vdash bs(\neg(\rhd f;g)) = (empty \vee (finite \wedge (\neg(di\ (\rhd f)) \vee (\rhd f \frown (\neg(df(g) \frown skip))))))$
using *014 09 1 11 12 2 3 5 6 7 8*
by (*metis 000 inteq-reflection*)
show *?thesis* **using** *025 026* **by** (*metis FstImpFinite Prop10 inteq-reflection schop-d-def*)
qed

lemma *FstFstChopEqvFstChopFst*:
$\vdash \rhd(\rhd f;g) = \rhd f;\rhd g$
proof −
have *1*: $\vdash \rhd(\rhd f;g) = ((\rhd f;g) \wedge bs\ (\neg(\rhd f;g)))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash bs\ (\neg(\rhd f;g)) = (empty \vee (finite \wedge \neg(di(\rhd f))) \vee (\rhd f;bs(\neg g)))$
using *BsNotFstChop* **by** *auto*
hence *3*: $\vdash ((\rhd f;g) \wedge bs\ (\neg(\rhd f;g))) = ((\rhd f;g) \wedge (empty \vee (finite \wedge \neg(di(\rhd f))) \vee (\rhd f;bs(\neg g))))$
by *auto*
have *4*: $\vdash ((\rhd f;g) \wedge (empty \vee (finite \wedge \neg(di(\rhd f))) \vee (\rhd f;bs(\neg g)))) =$
$(((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge \neg(di(\rhd f)) \wedge finite) \vee ((\rhd f;g) \wedge (\rhd f;bs(\neg g))))$
by *auto*
have *5*: $\vdash \neg((\rhd f;g) \wedge \neg(di(\rhd f)) \wedge finite)$
using *ChopImpDi* **by** *fastforce*
hence *6*: $\vdash (((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge \neg(di(\rhd f)) \wedge finite) \vee ((\rhd f;g) \wedge (\rhd f;bs(\neg g)))) =$
$(((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge (\rhd f;bs(\neg g))))$
by *auto*
have *7*: $\vdash ((\rhd f;g) \wedge (\rhd f;(bs(\neg g)))) = ((\rhd f;(g \wedge (bs(\neg g)))))$
using *LFstAndDistrC* **by** *blast*
hence *8*: $\vdash (((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge (\rhd f;(bs(\neg g))))) =$
$(((\rhd f;g) \wedge empty) \vee ((\rhd f;(g \wedge (bs(\neg g))))))$
by *auto*
have *9*: $\vdash (((\rhd f;g) \wedge empty) \vee ((\rhd f;(g \wedge (bs(\neg g)))))) = (((\rhd f;g) \wedge empty) \vee \rhd f;\rhd g)$
by (*simp add*: *first-d-def*)
have *10*: $\vdash ((\rhd f;g) \wedge empty) = ((\rhd f;\rhd g) \wedge empty)$
using *FstChopEmptyEqvFstChopFstEmpty*
by (*metis ChopEmptyAndEmpty SChopEmptyAndEmpty inteq-reflection*)
hence *11*: $\vdash (((\rhd f;g) \wedge empty) \vee \rhd f;\rhd g) = (((\rhd f;\rhd g) \wedge empty) \vee \rhd f;\rhd g)$

by *auto*
have *12*: $\vdash (((\triangleright f;\triangleright g) \wedge empty) \vee \triangleright f;\triangleright g) = \triangleright f;\triangleright g$
by *auto*
from *1 3 4 6 8 9 11 12* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *FstFixFst*:
$\vdash \triangleright(\triangleright f) = \triangleright f$
proof −
have *1*: $\vdash \triangleright f = (\triangleright f);empty$ **using** *ChopEmpty* **by** (*metis int-eq*)
hence *2*: $\vdash \triangleright(\ \triangleright f\ ) = \triangleright((\triangleright f);empty)$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \triangleright((\triangleright f);empty) = \triangleright f;\triangleright empty$ **using** *FstFstChopEqvFstChopFst* **by** *auto*
have *4*: $\vdash \triangleright f;\triangleright empty = \triangleright f;empty$ **using** *FstEmpty* **using** *RightChopEqvChop* **by** *blast*
have *5*: $\vdash \triangleright f;empty = \triangleright f$ **using** *ChopEmpty* **by** *blast*
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstCSEqvEmpty*:
$\vdash \triangleright(f^\star) = empty$
proof −
have *1*: $\vdash \triangleright(f^\star) = \triangleright(empty \vee ((f \wedge more);f^\star))$ **using** *ChopstarEqv FstEqvRule* **by** *blast*
from *1* **show** *?thesis* **using** *FstEmptyOrEqvEmpty* **by** *fastforce*
qed

lemma *FstIterFixFst*:
$\vdash fpower\ (\triangleright\ f)\ n = \triangleright(fpower\ (\triangleright\ f)\ n)$
proof
(*induct n*)
case *0*
then show *?case*
proof −
have *1*: $\vdash fpower\ (\triangleright\ f)\ 0 = empty$
by (*simp add*: *fpower-d-def*)
have *2*: $\vdash empty = \triangleright\ empty$ **using** *FstEmpty* **by** *auto*
have *3*: $\vdash \triangleright\ empty = \triangleright(fpower\ (\triangleright f)\ 0)$
by (*metis 1 2 inteq-reflection*)
from *1 2 3* **show** *?thesis*
by *fastforce*
qed
next
case (*Suc n*)
then show *?case*
proof −
have *4*: $\vdash (fpower\ (\triangleright\ f)\ (Suc\ n)) = (\triangleright\ f)\ ;(fpower\ (\triangleright f)\ n)$
by (*metis FstImpFinite LeftChopEqvChop Prop10 Prop11 fpower-d-def wpow-Suc*)
have *5*: $\vdash (\triangleright\ f)\ ;(fpower\ (\triangleright f)\ n) = (\triangleright\ f)\ ;\triangleright\ (fpower\ (\triangleright\ f)\ n)$
using *RightChopEqvChop Suc.hyps* **by** *blast*
have *6*: $\vdash (\triangleright\ f)\ ;\triangleright\ (fpower\ (\triangleright\ f)\ n) = \triangleright(\triangleright f;(fpower\ (\triangleright\ f)\ n))$
using *FstFstChopEqvFstChopFst* **by** *fastforce*
have *7*: $\vdash \triangleright(\triangleright f;(fpower\ (\triangleright\ f)\ n)) = \triangleright(fpower\ (\triangleright\ f)\ (Suc\ n))$

by (*metis 4 6 inteq-reflection*)
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *DsImpNotFst*:
$\vdash$ *ds f* $\longrightarrow$ $(\neg(\triangleright f))$
proof −
have *1*: $\vdash$ $(ds\ f \wedge \triangleright f) = (ds\ f \wedge (f \wedge bs\ (\neg\ f)))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash$ $(ds\ f \wedge (f \wedge bs\ (\neg\ f))) = (ds\ f \wedge f \wedge \neg(ds\ f))$ **using** *NotDsEqvBsNot* **by** *fastforce*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstLenAndEqvLenAnd*:
$\vdash \triangleright(len(k) \wedge f) = (len(k) \wedge f)$
proof −
have *1*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow ds\ (len(k))$
using *DsAndImpElimL* **by** *fastforce*
have *01*:$\vdash$ $ds(len(k) \wedge f) = (finite \longrightarrow di(len(k) \wedge f)^\frown skip)$
by (*simp add*: *DsDi*)
have *02*:$\vdash$ $len\ k \longrightarrow finite$
by (*simp add*: *len-k-finite*)
have *03*:$\vdash$ $di(len(k) \wedge f)^\frown skip \longrightarrow (di(len(k)))^\frown skip$
by (*simp add*: *DiAndA LeftSChopImpSChop*)
hence *2*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (di(len(k)))^\frown skip$
using *01 02 03* **by** *fastforce*
hence *3*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow ((len(k)^\frown \# True))^\frown skip$
unfolding *di-d-def*
by (*metis Prop10 inteq-reflection len-k-finite schop-d-def*)
hence *4*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k)^\frown(\# True^\frown skip))$
using *SChopAssoc* **by** *fastforce*
hence *5*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k)^\frown(skip^\frown finite))$
by (*metis Prop10 SkipFiniteEqvFiniteSkip WPowerstar-ext WPowerstar-skip-finite int-simps*(*17*) *inteq-reflection schop-d-def*)
hence *6*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k)^\frown(skip^\frown finite)) \wedge len(k)$
by *auto*
hence *7*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k)^\frown(skip^\frown finite)) \wedge len(k)^\frown empty$
by (*metis ChopEmpty Prop12 inteq-reflection len-k-finite lift-imp-trans schop-d-def*)
hence *8*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k)^\frown((skip^\frown finite) \wedge empty))$
by (*metis LenSChopAnd Prop11 lift-imp-trans*)
have *08*: $\vdash$ $\neg((skip^\frown finite) \wedge empty)$
by (*metis ChopEmptyAndEmpty FmoreEqvSkipChopFinite NotFmoreAndEmpty SChopEmptyAndEmpty inteq-reflection lift-and-com*)
have *09*:$\vdash$ $((skip^\frown finite) \wedge empty) = \# False$
using *08* **by** (*simp add*: *Prop11*)
have *010*:$\vdash$ $len(k)^\frown((skip^\frown finite) \wedge empty) = \# False$
by (*metis 09 SChopRightFalse inteq-reflection*)
have *9*: $\vdash$ $\neg(len(k)^\frown((skip^\frown finite) \wedge empty))$
using *010* **by** *auto*
have *10*: $\vdash$ $len(k) \wedge f \longrightarrow \neg(ds(len(k) \wedge f))$

using *8 9* **by** *fastforce*
hence *11*: $\vdash$ *len*(*k*) $\wedge$ *f* $\longrightarrow$ *bs* ($\neg$(*len*(*k*)$\wedge$ *f*))
using *NotDsEqvBsNot* **by** *fastforce*
hence *12*: $\vdash$ *len*(*k*) $\wedge$ *f* $\longrightarrow$ (*len*(*k*) $\wedge$ *f*) $\wedge$ *bs* ($\neg$(*len*(*k*)$\wedge$ *f*))
by *auto*
hence *13*: $\vdash$ *len*(*k*) $\wedge$ *f* $\longrightarrow$ $\triangleright$(*len*(*k*) $\wedge$ *f*)
by (*simp add*: *first-d-def*)
have *14*: $\vdash$ $\triangleright$(*len*(*k*) $\wedge$ *f*) $\longrightarrow$ *len*(*k*) $\wedge$ *f*
by (*auto simp add*: *first-d-def*)
from *13 14* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndElimL*:
$\vdash$ $\triangleright$*f* $\longrightarrow$ *f*
by (*auto simp add*: *first-d-def*)

lemma *FstImpNotDiChopSkip*:
$\vdash$ $\triangleright$*f* $\longrightarrow$ $\neg$(*di f*$\frown$*skip*)
proof −
have *1*: $\vdash$ $\triangleright$*f* $\longrightarrow$ *bs* ($\neg$ *f*) **by** (*auto simp add*: *first-d-def*)
hence *2*: $\vdash$ $\triangleright$*f* $\longrightarrow$ $\neg$(*ds f*) **using** *NotDsEqvBsNot* **by** *fastforce*
have *3*: $\vdash$ *ds f* = (*finite* $\longrightarrow$ *di f* $\frown$ *skip*) **using** *DsDi* **by** *blast*
hence *4*: $\vdash$ ($\neg$(*ds f*)) = (*finite* $\wedge$ $\neg$(*di f*$\frown$*skip*)) **by** *auto*
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *FstImpNotDiChopSkipB*:
$\vdash$ $\triangleright$*f* $\longrightarrow$ $\neg$(*di* (*f*$\frown$*skip*))
proof −
have *1*: $\vdash$ $\triangleright$*f* $\longrightarrow$ *bs* ($\neg$ *f*)
by (*auto simp add*: *first-d-def*)
hence *2*: $\vdash$ $\triangleright$*f* $\longrightarrow$ $\neg$(*ds f*)
using *NotDsEqvBsNot* **by** *fastforce*
have *3*: $\vdash$ *ds f* = (*finite* $\longrightarrow$ *di f* $\frown$ *skip*)
using *DsDi* **by** *blast*
have *4*: $\vdash$ *di f* $\frown$ *skip* = (*f*$\frown$#*True*)$\frown$*skip*
by (*metis ChopTrueAndFiniteEqvAndFiniteChopFinite SChopFiniteEqvSChopTrueAndFinite di-d-def int-simps*(*1*) *inteq-reflection schop-d-def*)
have *5*: $\vdash$(*f*$\frown$#*True*)$\frown$*skip* = *f*$\frown$(#*True*$\frown$*skip*)
by (*meson Prop11 SChopAssoc*)
have *6*: $\vdash$ *f*$\frown$(#*True*$\frown$*skip*) = *f*$\frown$(*skip*$\frown$*finite*)
by (*metis ChopTrueAndFiniteEqvAndFiniteChopFinite DiSkipEqvMore DiamondSChopdef FmoreEqvSkipChopFinite RightSChopEqvSChop SkipFiniteEqvFiniteSkip di-d-def fmore-d-def inteq-reflection schop-d-def sometimes-d-def*)
have *7*: $\vdash$ *f*$\frown$(*skip*$\frown$*finite*) = (*f*$\frown$*skip*)$\frown$*finite*
using *SChopAssoc* **by** *blast*
have *8*: $\vdash$ (*f*$\frown$*skip*)$\frown$*finite* = (*di*(*f*$\frown$*skip*) $\wedge$ *finite*)
by (*metis ChopTrueAndFiniteEqvAndFiniteChopFinite Prop11 di-d-def schop-d-def*)
have *9*: $\vdash$ ($\neg$(*ds f*)) = ($\neg$(*di*(*f*$\frown$*skip*)) $\wedge$ *finite*)
using *3 4 5 6 7 8* **by** *fastforce*

from *2 9* **show** *?thesis* **by** *fastforce*
qed

lemma *FstImpDiEqv*:
$\vdash \triangleright f \longrightarrow (di\ f = f)$
proof −
have *1*: $\vdash \triangleright f \longrightarrow \neg(di\ f \frown skip)$ **using** *FstImpNotDiChopSkip* **by** *blast*
have *2*: $\vdash di\ f \wedge finite \longrightarrow f \vee (di\ f \frown skip)$ **using** *DiEqvOrDiChopSkipB*
using *DiFiniteEqv* **by** *fastforce*
have *3*: $\vdash \triangleright f \wedge di\ f \longrightarrow (f \vee (di\ f \frown skip)) \wedge \neg(di\ f \frown skip)$ **using** *1 2*
using *FstAndElimL* **by** *fastforce*
have *4*: $\vdash ((f \vee (di\ f \frown skip)) \wedge \neg(di\ f \frown skip)) = (f \wedge \neg(di\ f \frown skip))$ **by** *auto*
have *5*: $\vdash \triangleright f \wedge di\ f \longrightarrow f \wedge \neg(di\ f \frown skip)$ **using** *3 4* **by** *fastforce*
hence *6*: $\vdash \triangleright f \wedge di\ f \longrightarrow f$ **by** *fastforce*
hence *7*: $\vdash \triangleright f \longrightarrow (di\ f \longrightarrow f)$ **using** *FstAndElimL* **by** *fastforce*
have *8*: $\vdash f \longrightarrow di\ f$ **using** *DiIntro* **by** *auto*
hence *9*: $\vdash \triangleright f \longrightarrow (f \longrightarrow (di\ f))$ **by** *auto*
from *7 9* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiFstAndEqvFstAnd*:
$\vdash (\triangleright f \wedge\ di(\triangleright f \wedge g)) = (\triangleright f \wedge g)$
proof −
have *1*: $\vdash \triangleright f \wedge\ di(\triangleright f \wedge g) \longrightarrow \triangleright f$
by *auto*
have *2*: $\vdash \triangleright f \wedge\ di(\triangleright f \wedge g) \longrightarrow di(\triangleright f \wedge g)$
by *auto*
have *3*: $\vdash finite \longrightarrow di(\triangleright f \wedge g) = ((\triangleright f \wedge g) \vee di((\triangleright f \wedge g) \frown skip))$
using *DiEqvOrDiChopSkipA*
by (*metis* (*no-types*, *lifting*) *DiEqvDiFst FstFstAndEqvFstAnd FstImpFinite Prop10 inteq-reflection schop-d-def*)
have *4*: $\vdash finite \longrightarrow di((\triangleright f \wedge g) \frown skip) = ((\triangleright f \wedge g) \frown skip) \frown finite$
unfolding *di-d-def schop-d-def*
using *ChopTrueAndFiniteEqvAndFiniteChopFinite* **by** *fastforce*
have *04*:$\vdash \triangleright f \longrightarrow finite$
by (*simp add*: *FstImpFinite*)
have *5*: $\vdash \triangleright f \wedge\ di(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g) \vee ((\triangleright f \wedge g) \frown skip) \frown finite$
using *2 3 4 04* **by** *fastforce*
have *6*: $\vdash \triangleright f \wedge g \longrightarrow f$
using *FstAndElimL* **by** *fastforce*
hence *7*: $\vdash ((\triangleright f \wedge g) \frown skip) \frown finite \longrightarrow (f \frown skip) \frown finite$
by (*simp add*: *LeftSChopImpSChop*)
hence *8*: $\vdash ((\triangleright f \wedge g) \frown skip) \frown finite \longrightarrow di(f \frown skip)$
by (*metis AndChopA ChopImpDi lift-imp-trans schop-d-def*)
have *9*: $\vdash \triangleright f \longrightarrow \neg(di(f \frown skip))$
using *FstImpNotDiChopSkipB* **by** *blast*
have *10*: $\vdash \triangleright f \wedge\ di(\triangleright f \wedge g) \longrightarrow ((\triangleright f \wedge g) \vee di(f \frown skip))$
using *5 8* **by** *fastforce*
have *11*: $\vdash \triangleright f \wedge\ di(\triangleright f \wedge g) \longrightarrow \neg(di(f \frown skip)) \wedge ((\triangleright f \wedge g) \vee di(f \frown skip))$
using *9 10 1* **by** *fastforce*

have *12*: $\vdash (\neg(di(f \frown skip)) \wedge ((\rhd f \wedge g) \vee di(f \frown skip))) = (\neg(di(f \frown skip)) \wedge ((\rhd f \wedge g)))$
by *auto*
have *13*: $\vdash \rhd f \wedge \ di(\rhd f \wedge g) \longrightarrow (\rhd f \wedge g)$
using *11 12* **by** *auto*
have *14*: $\vdash (\rhd f \wedge g) \longrightarrow \rhd f$
by *auto*
hence *15*: $\vdash (\rhd f \wedge g) \longrightarrow di(\rhd f \wedge g\ )$
using *DiIntro* **by** *auto*
have *16*: $\vdash (\rhd f \wedge g) \longrightarrow \rhd f \wedge di(\rhd f \wedge g)$
using *14 15* **by** *auto*
from *13 16* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiImpBsNotAndDi*:
$\vdash (\rhd f \wedge di\ g) \longrightarrow (bs\ (\neg(di\ f \wedge g)))$
proof −
have *1*: $\vdash (\rhd f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds(di\ f \wedge g)$
by (*auto simp add*: *ds-d-def*)
hence *2*: $\vdash (\rhd f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds(di\ f)$
using *DsAndImp* **by** *fastforce*
hence *3*: $\vdash (\rhd f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow di(di\ f) \frown skip$
using *DsDi FstImpFinite* **by** *fastforce*
hence *4*: $\vdash (\rhd f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow di\ f \frown skip$
using *DiEqvDiDi* **by** (*metis int-eq*)
hence *5*: $\vdash (\rhd f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds\ f$
using *DsDi* **by** *fastforce*
hence *6*: $\vdash \ (\rhd f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow\neg\ (\rhd\ f)$
using *DsImpNotFst* **by** *fastforce*
from *6* **show** *?thesis* **by** *auto*
qed

lemma *FstFstOrEqvFstOrL*:
$\vdash \rhd(\rhd f \vee g) = \rhd(f \vee g)$
proof −
have *1*: $\vdash \rhd(f \vee g) = ((f \vee g) \wedge bs\ (\neg(f \vee g)))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash \ (\neg(f \vee g)) = \ (\neg f \wedge \neg\ g)$
by *auto*
hence *3*: $\vdash \ bs(\neg(f \vee g)) = \ bs\ (\neg f \wedge \neg\ g)$
using *BsEqvRule* **by** *blast*
have *4*: $\vdash bs\ (\neg f \wedge \neg\ g) = (bs\ (\neg\ f) \wedge bs\ (\neg\ g))$
using *BsAndEqv* **by** *fastforce*
hence *5*: $\vdash ((f \vee g) \wedge bs(\neg(f \vee g))) = \ ((f \vee g) \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g))$
using *4 3* **by** *fastforce*
have *6*: $\vdash ((f \vee g) \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g)) =$
$(((f \wedge bs\ (\neg\ f)) \vee (g \wedge bs\ (\neg\ f))) \wedge bs\ (\neg\ g))$
by *auto*
have *7*: $\vdash (((f \wedge bs\ (\neg\ f)) \vee (g \wedge bs\ (\neg\ f))) \wedge bs\ (\neg\ g)) =$
$((\rhd f \vee (g \wedge bs\ (\neg\ f))) \wedge bs\ (\neg\ g))$
by (*simp add*: *first-d-def*)

have *8*: $\vdash ((\triangleright f \vee (g \wedge bs\ (\neg\ f))) \wedge bs\ (\neg\ g)) =$
$(((\triangleright f \vee g) \wedge (\triangleright f \vee bs\ (\neg f))) \wedge bs\ (\neg\ g))$
by *auto*
have *9*: $\vdash (((\triangleright f \vee g) \wedge (\triangleright f \vee bs\ (\neg f))) \wedge bs\ (\neg\ g)) =$
$(((\triangleright f \vee g) \wedge ((f \wedge bs\ (\neg\ f)) \vee bs\ (\neg f))) \wedge bs\ (\neg\ g))$
by (*simp add*: *first-d-def*)
have *10*: $\vdash (((\triangleright f \vee g) \wedge ((f \wedge bs\ (\neg\ f)) \vee bs\ (\neg f))) \wedge bs\ (\neg\ g)) =$
$((\triangleright f \vee g) \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g))$
by *auto*
have *11*: $\vdash ((\triangleright f \vee g) \wedge bs\ (\neg\ f) \wedge bs\ (\neg\ g)) =$
$((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs\ (\neg\ g))$
using *BsNotFstEqvBsNot* **by** *fastforce*
have *12*: $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs\ (\neg\ g)) =$
$((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f) \wedge \neg\ g))$
using *BsAndEqv* **by** *fastforce*
have *13*: $\vdash\ (\neg(\triangleright f) \wedge \neg\ g) = (\neg(\triangleright f \vee g))$
by *auto*
hence *14*: $\vdash bs\ (\neg(\triangleright f) \wedge \neg\ g) = bs\ (\neg(\triangleright f \vee g))$
using *BsEqvRule* **by** *blast*
hence *15*: $\vdash ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f) \wedge \neg\ g)) = ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f \vee g)))$
by *auto*
have *16*: $\vdash\ ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f \vee g))) = \triangleright(\triangleright f \vee g)$
by (*simp add*: *first-d-def*)
from *16 15 12 11 10 9 8 7 6 5 1* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *FstFstOrEqvFstOrR*:
$\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$
proof −
have *1*: $\vdash (f \vee \triangleright g) = (\triangleright g \vee f)$ **by** *auto*
hence *2*: $\vdash \triangleright\ (f \vee \triangleright g) = \triangleright(\triangleright g \vee f)$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \triangleright(\triangleright g \vee f) = \triangleright(g \vee f)$ **using** *FstFstOrEqvFstOrL* **by** *blast*
have *4*: $\vdash (g \vee f) = (f \vee g)$ **by** *auto*
hence *5*: $\vdash \triangleright(g \vee f) = \triangleright(f \vee g)$ **using** *FstEqvRule* **by** *blast*
from *2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstFstOrEqvFstOr*:
$\vdash \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee g)$
proof −
have *1*: $\vdash\ \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee \triangleright g)$ **using** *FstFstOrEqvFstOrL* **by** *blast*
have *2*: $\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$ **using** *FstFstOrEqvFstOrR* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstLenEqvLen*:
$\vdash \triangleright(\ len(k)) = len(k)$
proof −
have *1*: $\vdash \triangleright(\ len(k) \wedge \#True) = (len(k) \wedge \#True)$ **using** *FstLenAndEqvLenAnd* **by** *blast*
have *2*: $\vdash (len(k) \wedge \#True) = len(k)$ **by** *auto*

hence *3*: $\vdash \rhd(\ len(k) \wedge \#True) = \rhd(len(k))$ **using** *FstEqvRule* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *FstSkip*:
$\vdash \rhd\ skip = skip$
proof −
have *1*: $\vdash skip = len(1)$ **using** *LenOneEqvSkip* **by** *fastforce*
hence *2*: $\vdash \rhd skip = \rhd\ (len(1))$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \rhd(\ len(1)) = len(1)$ **using** *FstLenEqvLen* **by** *blast*
from *1 2 3* **show** *?thesis* **using** *LenOneEqvSkip* **by** *fastforce*
qed

lemma *HaltStateEqvFstFinState*:
$\vdash (halt\ (init\ w) \wedge finite) = \rhd\ (fin\ (init\ w))$
proof −
have *1*: $\vdash halt\ (init\ w) = \Box(empty = (init\ w))$ **by** (*simp add*: *halt-d-def*)
have *21*: $\vdash (empty = (init\ w)) = (((empty \longrightarrow (init\ w)) \wedge ((init\ w) \longrightarrow empty)))$
by *auto*
hence *2*: $\vdash \Box(empty = (init\ w)) = (\Box((empty \longrightarrow (init\ w)) \wedge ((init\ w) \longrightarrow empty)))$
by (*simp add*: *BoxEqvBox*)
have *3*: $\vdash (\Box((empty \longrightarrow (init\ w)) \wedge ((init\ w) \longrightarrow empty))) =$
$(\Box((empty \longrightarrow (init\ w))) \wedge \Box((init\ w) \longrightarrow empty))$
by (*metis 21 BoxAndBoxEqvBoxRule int-eq*)
have *4*: $\vdash ((init\ w) \longrightarrow empty) = (more \longrightarrow \neg(init\ w))$
by (*auto simp add*: *empty-d-def*)
hence *5*: $\vdash \Box\ ((init\ w) \longrightarrow empty) = \Box\ (more \longrightarrow \neg(init\ w))$ **using** *BoxEqvBox* **by** *blast*
have *6*: $\vdash (\Box\ (more \longrightarrow \neg(init\ w)) \wedge finite) = bs(\neg(fin(init\ w)))$ **using** *BoxMoreStateEqvBsFinState* **by** *blast*
have *7*: $\vdash \Box((empty \longrightarrow (init\ w))) = fin(init\ w)$ **by** (*simp add*: *fin-d-def*)
have *8*: $\vdash\ (\Box((empty \longrightarrow (init\ w))) \wedge \Box((init\ w) \longrightarrow empty) \wedge finite) =$
$(fin(init\ w) \wedge bs(\neg(fin(init\ w))))$ **using** *5 6 7* **by** *fastforce*
from *1 2 3 8* **show** *?thesis* **unfolding** *first-d-def*
by *fastforce*
qed

lemma *FstLenEqvLenFst*:
$\vdash \rhd(len\ k\ ;\ f) = len\ k\ ;\ \rhd\ f$
proof −
have *1*: $\vdash len\ k\ ;\ f = \rhd(len\ k)\ ;\ f$ **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*
have *2*: $\vdash \rhd(len\ k\ ;\ f) = \rhd\ (\rhd(len\ k)\ ;\ f)$ **using** *1 FstEqvRule* **by** *blast*
have *3*: $\vdash \rhd\ (\rhd(len\ k)\ ;\ f) = \rhd(len\ k)\ ;\ \rhd f$ **using** *FstFstChopEqvFstChopFst* **by** *blast*
have *4*: $\vdash \rhd(len\ k)\ ;\ \rhd f = len\ k\ ;\ \rhd f$ **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *FstNextEqvNextFst*:
$\vdash \rhd(\bigcirc f) = \bigcirc(\ \rhd f)$
proof −

have *1*: $\vdash \rhd(\bigcirc\ f) = \rhd(skip\ ;\ f)$ **using** *FstEqvRule* **by** (*simp add*: *next-d-def*)
have *2*: $\vdash skip\ ;\ f = \rhd skip\ ;\ f$ **using** *FstSkip* **using** *LeftChopEqvChop* **by** *fastforce*
have *3*: $\vdash \rhd(skip\ ;\ f) = \rhd\ (\rhd skip\ ;\ f)$ **using** *2 FstEqvRule LeftChopEqvChop* **by** *blast*
have *4*: $\vdash \rhd\ (\rhd skip\ ;\ f) = \rhd skip\ ;\ \rhd f$ **using** *3 FstFstChopEqvFstChopFst* **by** *blast*
have *5*: $\vdash \rhd skip\ ;\ \rhd f = skip\ ;\ \rhd f$ **using** *4 FstSkip LeftChopEqvChop* **by** *blast*
have *6*: $\vdash skip\ ;\ \rhd f\ =\ \bigcirc(\ \rhd f)$ **by** (*simp add*: *next-d-def*)
from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstDiamondStateEqvHalt*:
$\vdash \rhd\ (\Diamond\ (init\ w)) = (halt\ (init\ w) \wedge finite)$
proof −
have *1*: $\vdash \Diamond\ (init\ w) = \Diamond\ ((init\ w) \wedge \#True)$ **by** *simp*
have *2*: $\vdash fin\ (init\ w) \frown \#True = \Diamond\ ((init\ w) \wedge \#True)$ **using** *1 FinChopEqvDiamond*
by (*metis schop-d-def*)
have *3*: $\vdash fin\ (init\ w) \frown \#True = df\ (fin\ (init\ w))$ **unfolding** *df-d-def* **by** *simp*
have *4*: $\vdash (\Diamond\ (init\ w)) = (df\ (fin\ (init\ w)))$ **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash \rhd\ (\Diamond\ (init\ w)) = \rhd\ (df\ (fin\ (init\ w)))$ **using** *4 FstEqvRule* **by** *blast*
hence *6*: $\vdash \rhd\ (\Diamond\ (init\ w)) = \rhd\ (fin\ (init\ w))$ **using** *FstDfEqvFst* **by** *fastforce*
hence *7*: $\vdash \rhd\ (\Diamond\ (init\ w)) = (halt\ (init\ w) \wedge finite)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
from *7* **show** *?thesis* **by** *simp*
qed

lemma *DiAndFiniteEqvDiFst*:
$\vdash di\ (f \wedge finite) = di(\rhd\ f)$
proof −
have *1*: $\vdash (\rhd\ f \wedge finite) = \rhd\ f$
by (*meson FstImpFinite Prop10 Prop11*)
have *2*: $\vdash (f \wedge finite);\#True = (\rhd\ f \wedge finite);\#True$
using *DfEqvDfFst FstImpFinite* **unfolding** *df-d-def schop-d-def* **by** *auto*
show *?thesis*
by (*metis 1 2 di-d-def inteq-reflection*)
qed

lemma *FstEqvFstAndFinite*:
$\vdash \rhd\ f = \rhd\ (f \wedge finite)$
by (*metis FstDfEqvFst FstDiEqvFst di-d-def inteq-reflection itl-def*(*15*) *schop-d-def*)

lemma *FstDiAndDiEqvFstDfAndDf*:
$\vdash \rhd(di\ f \wedge di\ g) = \rhd(df\ f \wedge df\ g)$
proof −
have *1*: $\vdash\ \rhd(di\ f \wedge di\ g) = \rhd\ ((di\ f \wedge di\ g) \wedge finite)$
by (*simp add*: *FstEqvFstAndFinite*)
have *2*: $\vdash ((di\ f \wedge di\ g) \wedge finite) = ((df\ f \wedge df\ g) \wedge finite)$
using *DiAndFiniteEqvDfAndFinite*[*of f*] *DiAndFiniteEqvDfAndFinite*[*of g*]
by *fastforce*
have *3*: $\vdash \rhd((di\ f \wedge di\ g) \wedge finite) = \rhd((df\ f \wedge df\ g) \wedge finite)$
by (*simp add*: *2 FstEqvRule*)

have *4*: $\vdash \rhd((df\ f \wedge df\ g) \wedge finite) = \rhd(df\ f \wedge df\ g)$
by (*meson FstEqvFstAndFinite Prop11*)
show *?thesis*
by (*metis 1 3 4 inteq-reflection*)
qed

lemma *FstBoxStateEqvStateAndEmpty*:
$\vdash \rhd\ (\Box\ (init\ w)) = ((init\ w) \wedge empty)$
proof −
have *1*: $\vdash ((init\ w) \wedge (\Box\ (init\ w))^\star) = \Box\ (init\ w)$
using *BoxCSEqvBox* **by** *blast*
have *2*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge (\Box\ (init\ w))^\star)$
using *1* **by** *auto*
hence *3*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge (\Box\ (init\ w))^\star)$
by *blast*
have *4*: $\vdash ((init\ w) \wedge empty)\ ;\ (\Box\ (init\ w))^\star = ((init\ w) \wedge (\Box\ (init\ w))^\star)$
using *StateAndEmptyChop* **by** *blast*
have *5*: $\vdash ((init\ w) \wedge (\Box\ (init\ w))^\star)\ =\ ((init\ w) \wedge empty)\ ;\ (\Box\ (init\ w))^\star$
using *4* **by** *fastforce*
have *6*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge empty)\ ;\ (\Box\ (init\ w))^\star$
using *3 5* **by** *fastforce*
have *7*: $\vdash ((init\ w) \wedge empty)\ ;\ (\Box\ (init\ w))^\star = \rhd\ (init\ w)\ ;\ (\Box\ (init\ w))^\star$
using *FstState* **by** (*metis AndChopCommute int-eq*)
have *8*: $\vdash \Box\ (init\ w) = \rhd\ (init\ w)\ ;\ (\Box\ (init\ w))^\star$
using *6 7* **by** *fastforce*
have *9*: $\vdash \rhd\ (\Box\ (init\ w)) = \rhd\ (\rhd\ (init\ w)\ ;\ (\Box\ (init\ w))^\star)$
using *8 FstEqvRule* **by** *blast*
have *10*: $\vdash \rhd\ (\rhd\ (init\ w)\ ;\ (\Box\ (init\ w))^\star) = \rhd\ (init\ w)\ ;\ \rhd\ ((\Box\ (init\ w))^\star)$
using *FstFstChopEqvFstChopFst* **by** *blast*
have *11*: $\vdash \rhd\ (init\ w)\ ;\ \rhd\ ((\Box\ (init\ w))^\star) = \rhd\ (init\ w)\ ;\ empty$
using *RightChopEqvChop FstCSEqvEmpty* **by** *blast*
have *12*: $\vdash \rhd\ (init\ w)\ ;\ empty = \rhd\ (init\ w)$
using *RightChopEqvChop ChopEmpty* **by** *blast*
have *13*: $\vdash \rhd\ (init\ w) = ((init\ w) \wedge empty)$
using *FstState* **by** *fastforce*
from *9 10 11 12 13* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndFstStarEqvFst*:
$\vdash (schopstar\ (\rhd f) \wedge (\rhd f)) = \rhd f$
proof −
have *1*: $\vdash (schopstar\ (\rhd f)) = (empty \vee (\rhd f) \frown (schopstar\ (\rhd f)))$
by (*meson Prop11 SChopstar-unfoldl-eq*)
have *2*: $\vdash ((schopstar\ (\rhd f)) \wedge \rhd f) = ((empty \vee (\rhd f) \frown (schopstar\ (\rhd f))) \wedge \rhd f)$
using *1* **by** *fastforce*
have *3*: $\vdash ((empty \vee (\rhd f) \frown (schopstar\ (\rhd f))) \wedge \rhd f) =$
$((empty \wedge \rhd f) \vee ((\rhd f) \frown (schopstar\ (\rhd f)) \wedge \rhd f))$
by *auto*

have *4*: $\vdash ((schopstar\ (\triangleright f)) \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee ((\triangleright f) \frown (schopstar\ (\triangleright f)) \wedge \triangleright f))$
using *2 3* **by** *fastforce*
have *5*: $\vdash ((\triangleright f) \frown (schopstar\ (\triangleright f)) \wedge \triangleright f) = ((\triangleright f) \frown (schopstar\ (\triangleright f)) \wedge \triangleright f \frown empty)$
by (*metis ChopEmpty FstImpFinite Prop10 inteq-reflection schop-d-def*)
have *6*: $\vdash ((\triangleright f) \frown (schopstar\ (\triangleright f)) \wedge \triangleright f \frown empty) = (\triangleright f) \frown ((schopstar\ (\triangleright f)) \wedge empty)$
by (*simp add*: *LFstAndDistrB schop-d-def*)
have *7*: $\vdash (schopstar\ (\triangleright f) \wedge empty) = empty$
using *SChopstar-imp-empty* **by** *fastforce*

have *8*: $\vdash (\triangleright f) \frown (schopstar\ (\triangleright f) \wedge empty) = \triangleright f$
using *7 ChopEmpty*
by (*metis FstImpFinite Prop11 Prop12 inteq-reflection schop-d-def*)
have *9*: $\vdash ((\triangleright f) \frown (schopstar\ (\triangleright f)) \wedge \triangleright f) = \triangleright f$
using *5 6 8* **by** *fastforce*
have *10*: $\vdash (schopstar\ (\triangleright f) \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee \triangleright f)$
using *4 9* **by** *fastforce*
have *11*: $\vdash ((empty \wedge \triangleright f) \vee \triangleright f) = \triangleright f$
by *auto*
have *12*: $\vdash (schopstar\ (\triangleright f) \wedge \triangleright f) = \triangleright f$
using *10 11* **by** *fastforce*
from *12* **show** *?thesis* **by** *auto*
qed

lemma *HaltStateEqvFstHaltState*:
$\vdash (halt(init(w)) \wedge finite) = \triangleright(halt(init(w)))$
proof −
have *1*: $\vdash (halt\ (init\ w) \wedge finite) = \triangleright\ (fin\ (init\ w))$
by (*simp add*: *HaltStateEqvFstFinState*)
have *2*: $\vdash \triangleright\ (fin\ (init\ w)) = \triangleright\ (\triangleright\ (fin\ (init\ w)))$
using *FstEqvRule FstFixFst* **by** *fastforce*
have *3*: $\vdash \triangleright\ (\triangleright\ (fin\ (init\ w))) = \triangleright(halt(init(w)) \wedge finite)$
using *FstEqvRule HaltStateEqvFstFinState* **by** (*metis inteq-reflection*)
have *4*: $\vdash \triangleright(halt(init(w)) \wedge finite) = \triangleright(halt(init(w)))$
by (*metis FstDfEqvFst FstDiEqvFst Prop04 di-d-def itl-def*(*15*) *schop-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DiHaltAndDiHaltAndEqvDiHaltAndAnd*:
$\vdash (df(halt\ (init\ w) \wedge f) \wedge df(halt\ (init\ w) \wedge g)) = df(halt\ (init\ w) \wedge f \wedge g)$
proof −

have *01*: $\vdash (halt\ (init\ w) \wedge f \wedge finite) = (\triangleright(fin\ (init\ w)) \wedge f \wedge finite)$
using *HaltStateEqvFstFinState* **by** *fastforce*
have *02*: $\vdash (halt\ (init\ w) \wedge g \wedge finite) = (\triangleright(fin\ (init\ w)) \wedge g \wedge finite)$
using *HaltStateEqvFstFinState* **by** *fastforce*
have *03*: $\vdash (halt\ (init\ w) \wedge f \wedge finite) \frown \# True = (\triangleright(fin\ (init\ w)) \wedge f \wedge finite) \frown \# True$
by (*simp add*: *01 LeftSChopEqvSChop*)
have *04*: $\vdash (halt\ (init\ w) \wedge g \wedge finite) \frown \# True = (\triangleright(fin\ (init\ w)) \wedge g \wedge finite) \frown \# True$
by (*simp add*: *02 LeftSChopEqvSChop*)

have *05*: $\vdash (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f \wedge \mathit{finite})^\frown \# \mathit{True} = (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f)^\frown \# \mathit{True}$
by (*metis* (*no-types*, *lifting*) *FstFstAndEqvFstAnd FstImpFinite LeftSChopEqvSChop Prop11 Prop12 inteq-reflection*)
have *06*: $\vdash (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge g \wedge \mathit{finite})^\frown \# \mathit{True} = (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge g)^\frown \# \mathit{True}$
by (*metis* (*no-types*, *lifting*) *FstFstAndEqvFstAnd FstImpFinite LeftSChopEqvSChop Prop11 Prop12 inteq-reflection*)
have *07*: $\vdash (\mathit{halt}\ (\mathit{init}\ w) \wedge f \wedge \mathit{finite})^\frown \# \mathit{True} = (\mathit{halt}\ (\mathit{init}\ w) \wedge f)^\frown \# \mathit{True}$
unfolding *schop-d-def*
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop Prop11 Prop12 lift-and-com*)
have *08*: $\vdash (\mathit{halt}\ (\mathit{init}\ w) \wedge g \wedge \mathit{finite})^\frown \# \mathit{True} = (\mathit{halt}\ (\mathit{init}\ w) \wedge g)^\frown \# \mathit{True}$
unfolding *schop-d-def*
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop Prop11 Prop12 lift-and-com*)
have *1*: $\vdash (\mathit{df}(\mathit{halt}\ (\mathit{init}\ w) \wedge f) \wedge \mathit{df}(\mathit{halt}\ (\mathit{init}\ w) \wedge g)) =$
$(\mathit{df}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f) \wedge \mathit{df}\ (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge g))$
unfolding *df-d-def*
by (*metis 01 02 05 06 07 08 inteq-reflection lift-and-com*)
have *2*: $\vdash (\mathit{df}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f) \wedge \mathit{df}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge g)) =$
$\mathit{df}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f \wedge g)$
using *LFstAndDistrD*
by (*metis* (*no-types*, *lifting*) *FstFstAndEqvFstAnd FstImpFinite Prop10 df-d-def di-d-def inteq-reflection schop-d-def*)
have *08*: $\vdash\ (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f \wedge g) = (\mathit{halt}\ (\mathit{init}\ w) \wedge f \wedge g \wedge \mathit{finite})$
using *HaltStateEqvFstFinState* **by** *fastforce*
have *09*: $\vdash (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f \wedge g)^\frown \# \mathit{True} = (\mathit{halt}\ (\mathit{init}\ w) \wedge f \wedge g \wedge \mathit{finite})^\frown \# \mathit{True}$
by (*simp add*: *08 LeftSChopEqvSChop*)
have *010*: $\vdash (\mathit{halt}\ (\mathit{init}\ w) \wedge f \wedge g \wedge \mathit{finite})^\frown \# \mathit{True} = (\mathit{halt}\ (\mathit{init}\ w) \wedge f \wedge g)^\frown \# \mathit{True}$
unfolding *schop-d-def*
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop Prop11 Prop12 lift-and-com*)
have *3*: $\vdash \mathit{df}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \wedge f \wedge g)\ = \mathit{df}(\mathit{halt}\ (\mathit{init}\ w) \wedge f \wedge g)$
unfolding *df-d-def*
by (*metis 010 08 inteq-reflection*)
from *1 2 3* **show** *?thesis* **using** *int-eq* **by** *metis*
qed

lemma *counter-ex-lhs*:
$\vdash ((\triangleright(\mathit{len}(5)) \wedge \triangleright(\mathit{len}(2)))\ ;\ (\mathit{len}(5) \vee \mathit{len}(2))) = \#\mathit{False}$
proof −
have *1*: $\vdash ((\triangleright(\mathit{len}(5)) \wedge \triangleright(\mathit{len}(2)))\ ;\ (\mathit{len}(5) \vee \mathit{len}(2))) =$
$(\mathit{len}(5) \wedge \mathit{len}(2));\ (\mathit{len}(5) \vee \mathit{len}(2))$
by (*metis FstLenAndEqvLenAnd FstLenEqvLen LeftChopEqvChop inteq-reflection*)
have *2*: $\vdash (\mathit{len}(5) \wedge \mathit{len}(2)) = \#\mathit{False}$
by (*simp add*: *Valid-def len-defs*)
have *3*: $\vdash ((\mathit{len}(5) \wedge \mathit{len}(2));\ (\mathit{len}(5) \vee \mathit{len}(2))) = (\#\mathit{False};(\mathit{len}(5) \vee \mathit{len}(2)))$
by (*simp add*: *2 LeftChopEqvChop*)
have *4*: $\vdash (\#\mathit{False};(\mathit{len}(5) \vee \mathit{len}(2))) = \#\mathit{False}$
by (*simp add*: *Valid-def chop-defs*)
from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *counter-ex-rhs*:
$\vdash ((\triangleright\ (len(5))\ ;\ (len(5) \vee len(2))) \wedge (\triangleright\ (len(2))\ ;\ (len(5) \vee len(2)))) = len(7)$
proof −
have *1*: $\vdash (\triangleright\ (len(5))\ ;\ (len(5) \vee len(2))) =$
$len(5);(len(5) \vee len(2))$
using *FstLenEqvLen LeftChopEqvChop* **by** *blast*
have *2*: $\vdash (\triangleright\ (len(2))\ ;\ (len(5) \vee len(2))) =$
$len(2)\ ;(len(5) \vee len(2))$
using *FstLenEqvLen LeftChopEqvChop* **by** *blast*
have *3*: $\vdash len(5);(len(5) \vee len(2)) =$
$((len(5);len(5)) \vee (len(5);len(2)))$
by (*simp add*: *ChopOrEqv*)
have *4*: $\vdash ((len(5);len(5)) \vee (len(5);len(2))) =$
$(len(10) \vee len(7))$
using *LenEqvLenChopLen inteq-reflection* **by** *fastforce*
have *5*: $\vdash len(2)\ ;(len(5) \vee len(2)) =$
$((len(2);len(5)) \vee (len(2);len(2)))$
by (*simp add*: *ChopOrEqv*)
have *6*: $\vdash ((len(2);len(5)) \vee (len(2);len(2))) =$
$(len(7) \vee len(4))$
using *LenEqvLenChopLen inteq-reflection* **by** *fastforce*
have *7*: $\vdash ((len(10) \vee len(7)) \wedge (len(7) \vee len(4))) =$
$((len(7) \vee len(10)) \wedge (len(7) \vee len(4)))$
by *fastforce*
have *8*: $\vdash ((len(7) \vee len(10)) \wedge (len(7) \vee len(4))) =$
$(len(7) \vee (len(10) \wedge len(4)))$
by *fastforce*
have *9*: $\vdash (len(10) \wedge len(4)) = \#False$
by (*simp add*: *Valid-def len-defs*)
have *10* : $\vdash (len(7) \vee (len(10) \wedge len(4))) = len(7)$
using *9* **by** *auto*
have *11*: $\vdash ((\triangleright\ (len(5))\ ;\ (len(5) \vee len(2))) \wedge (\triangleright\ (len(2))\ ;\ (len(5) \vee len(2)))) =$
$(len(5);(len(5) \vee len(2)) \wedge len(2)\ ;(len(5) \vee len(2)))$
using *1 2* **by** *fastforce*
have *12*: $\vdash (len(5);(len(5) \vee len(2)) \wedge len(2)\ ;(len(5) \vee len(2))) = len(7)$
by (*metis 10 4 6 7 8 ChopOrEqv inteq-reflection*)
from *11 12* **show** *?thesis* **by** *fastforce*
qed

end

3.14 Monitors

theory *Monitor*
imports *First*

begin

The RV monitors language is introduced plus the algebraic properties of the monitor operators.

3.14.1 Syntax

datatype ('a ::*world*) *monitor* =
  mFIRST-d 'a *formula* ((*FIRST* -) [*84*] *83*)
| *mUPTO-d* 'a *monitor* 'a *monitor* ((- *UPTO* -) [*84,84*] *83*)
| *mTHRU-d* 'a *monitor* 'a *monitor* ((- *THRU* -) [*84,84*] *83*)
| *mTHEN-d* 'a *monitor* 'a *monitor* ((- *THEN* -) [*84,84*] *83*)
| *mWITH-d* 'a *monitor* 'a *formula* ((- *WITH* -) [*84,84*] *83*)

fun *MON* :: ('a::*world*) *monitor* ⇒ 'a *formula*
where (*MON* (*FIRST f*)) = *LIFT*(▷ *f*)
  | (*MON* (*a UPTO b*)) = *LIFT*(▷((*MON a*) ∨ (*MON b*)))
  | (*MON* (*a THRU b*)) = *LIFT*(▷(*di*(*MON a*) ∧ *di*(*MON b*)))
  | (*MON* (*a THEN b*)) = *LIFT*((*MON a*)⌢(*MON b*))
  | (*MON* (*a WITH f*)) = *LIFT*((*MON a*) ∧ *f*)

syntax
 -*MON* :: 'a *monitor* ⇒ *lift* ((ℳ -) [*80*] *80*)

translations
 -*MON* == *CONST MON*

definition *eq-d* :: ('a:: *world*) *monitor* ⇒ 'a *monitor* ⇒ *bool* ((- ≃ -) [*84,84*] *83*)
where
 eq-d a b ≡ (⊢ (ℳ *a*) = (ℳ *b*))

lemma *MonEqRefl*:
 a ≃ *a*
by (*simp add*: *eq-d-def*)

lemma *MonEqSym*:
 assumes *a* ≃ *b*
 shows *b* ≃ *a*
using *assms* **by** (*metis eq-d-def inteq-reflection*)

lemma *MonEqTrans*:
 assumes *a* ≃ *b*
  b ≃ *c*
 shows *a* ≃ *c*
using *assms*(*1*) *assms*(*2*) **by** (*metis eq-d-def inteq-reflection*)

lemma *MonEq*:
 (*a* ≃ *b*) = (⊢ (ℳ *a*) = (ℳ *b*))
by (*simp add*: *eq-d-def*)

lemma *MonEqSubstWith*:
 assumes *a* ≃ *b*

shows $(a\ WITH\ f) \simeq (b\ WITH\ f)$
using *assms* **by** (*metis MON.simps*(5) *eq-d-def inteq-reflection lift-and-com*)

lemma *MonEqSubstThen*:
assumes $a1 \simeq b1$
$a2 \simeq b2$
shows $(a1\ THEN\ a2) \simeq (b1\ THEN\ b2)$
using *assms* **by** (*simp add*: *SChopEqvSChop eq-d-def*)

lemma *MonEqSubstUpto*:
assumes $a1 \simeq b1$
$a2 \simeq b2$
shows $(a1\ UPTO\ a2) \simeq (b1\ UPTO\ b2)$
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *MON.simps*(2) *eq-d-def int-eq MonEqRefl*)

lemma *MonEqSubstThru*:
assumes $a1 \simeq b1$
$a2 \simeq b2$
shows $(a1\ THRU\ a2) \simeq (b1\ THRU\ b2)$
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *MON.simps*(3) *eq-d-def int-eq MonEqRefl*)

3.14.2 Derived Monitors

definition $HALT\text{-}d :: ('a :: world)\ formula \Rightarrow 'a\ monitor$
where $HALT\text{-}d\ w \equiv FIRST(LIFT(fin\ (init\ w)))$

definition $LEN\text{-}d ::\ nat \Rightarrow ('a ::world)\ monitor$
where
$LEN\text{-}d\ k \equiv FIRST\ (LIFT(len\ k))$

definition $EMPTY\text{-}d :: ('a:: world)\ monitor$
where
$EMPTY\text{-}d \equiv FIRST\ (LIFT(empty))$

definition $SKIP\text{-}d :: ('a:: world)\ monitor$
where
$SKIP\text{-}d \equiv FIRST\ (LIFT\ (skip))$

syntax
$-HALT\text{-}d :: lift \Rightarrow 'a\ monitor$ $((HALT\ -)\ [84]\ 83)$
$-LEN\text{-}d :: nat \Rightarrow 'a\ monitor$ $((LEN\ -)\ [84]\ 83)$
$-EMPTY\text{-}d :: 'a\ monitor$ $((EMPTY)\ )$
$-SKIP\text{-}d :: 'a\ monitor$ $((SKIP))$

syntax (*ASCII*)
$-HALT\text{-}d :: lift \Rightarrow 'a\ monitor$ $((HALT\ -)\ [84]\ 83)$
$-LEN\text{-}d :: nat \Rightarrow 'a\ monitor$ $((LEN\ -)\ [84]\ 83)$
$-EMPTY\text{-}d :: 'a\ monitor$ $((EMPTY))$
$-SKIP\text{-}d :: 'a\ monitor$ $((SKIP))$

translations
-HALT-d ⇌ CONST HALT-d
-LEN-d ⇌ CONST LEN-d
-EMPTY-d ⇌ CONST EMPTY-d
-SKIP-d ⇌ CONST SKIP-d

definition *GUARD-d* :: (*'a*::*world*) *formula* ⇒ *'a monitor*
where
GUARD-d w ≡ (*EMPTY WITH LIFT*(*init w*))

primrec *TIMES-d* :: (*'a* :: *world*) *monitor* ⇒ *nat* ⇒ *'a monitor*
where
TIMES-0 : *TIMES-d a 0* = *EMPTY*
| *TIMES-Suc*: *TIMES-d a* (*Suc k*) = (*a THEN* (*TIMES-d a k*))

syntax
-GUARD-d :: *lift* ⇒ *'a monitor* ((*GUARD -*) [*84*] *83*)
-TIMES-d :: [*'a monitor*,*nat*] ⇒ *'a monitor* ((*- TIMES -*) [*84*,*84*] *83*)

syntax (*ASCII*)
-GUARD-d :: *lift* ⇒ *'a monitor* ((*GUARD -*) [*84*] *83*)
-TIMES-d :: [*'a monitor*,*nat*] ⇒ *'a monitor* ((*- TIMES -*) [*84*,*84*] *83*)

translations
-GUARD-d ⇌ *CONST GUARD-d*
-TIMES-d ⇌ *CONST TIMES-d*

definition *FAIL-d* :: (*'a*:: *world*) *monitor*
where
FAIL-d ≡ *GUARD* (#*False*)

definition *ALWAYS-d* :: (*'a* :: *world*) *monitor* ⇒ *'a formula* ⇒ *'a monitor*
where
ALWAYS-d a w ≡ (*a WITH LIFT*((*bi* (*finite*⟶*fin* (*init w*)))))

definition *SOMETIME-d* :: (*'a* :: *world*) *monitor* ⇒ *'a formula* ⇒ *'a monitor*
where
SOMETIME-d a w ≡ (*a WITH LIFT*((*di* (*finite* ∧ *fin* (*init w*)))))

definition *LIMIT-d* :: (*'a* :: *world*) *formula* ⇒ *'a formula*
where
LIMIT-d f ≡ *LIFT*(*bs* (¬ *f*))

definition *UNTIL-d* :: (*'a* :: *world*) *formula* ⇒ *'a formula* ⇒ *'a monitor*
where
UNTIL-d w1 w2 ≡ (*HALT w2*) *WITH* (*LIFT*(*bm w1*))

syntax
-FAIL-d :: *'a monitor* (*FAIL*)
-ALWAYS-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *ALWAYS* -) [*84*,*84*] *83*)
-SOMETIME-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *SOMETIME* -) [*84*,*84*] *83*)
-LIMIT-d :: *lift* ⇒ *lift* ((*Limit* -) [*84*] *83*)
-UNTIL-d :: [*lift*,*lift*] ⇒ *'a monitor* ((- *UNTIL* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-FAIL-d :: *'a monitor* (*FAIL*)
-ALWAYS-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *ALWAYS* -) [*84*,*84*] *83*)
-SOMETIME-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *SOMETIME* -) [*84*,*84*] *83*)
-LIMIT-d :: *lift* ⇒ *lift* ((*Limit* -) [*84*] *83*)
-UNTIL-d :: [*lift*,*lift*] ⇒ *'a monitor* ((- *UNTIL* -) [*84*,*84*] *83*)

translations
-FAIL-d ⇌ *CONST FAIL-d*
-ALWAYS-d ⇌ *CONST ALWAYS-d*
-SOMETIME-d ⇌ *CONST SOMETIME-d*
-LIMIT-d ⇌ *CONST LIMIT-d*
-UNTIL-d ⇌ *CONST UNTIL-d*

definition *WITHIN-d* :: (*'a* :: *world*) *monitor* ⇒ *'a formula* ⇒ *'a monitor*
where
WITHIN-d a f ≡ (*a WITH LIFT*(*Limit f*))

syntax
-WITHIN-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *WITHIN* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-WITHIN-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *WITHIN* -) [*84*,*84*] *83*)

translations
-WITHIN-d ⇌ *CONST WITHIN-d*

definition *AND-d* :: (*'a* :: *world*) *monitor* ⇒ *'a monitor* ⇒ *'a monitor*
where
AND-d a b ≡ (*a WITH LIFT*(*M b*))

definition *ITERATE-d* :: (*'a* :: *world*) *monitor* ⇒ *'a monitor* ⇒ *'a monitor*
where
ITERATE-d a b ≡ (*a WITH* (*LIFT* (*schopstar* (*M b*))))

syntax
-AND-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *AND* -) [*84*,*84*] *83*)
-ITERATE-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *ITERATE* -) [*84*,*84*] *83*)

syntax (*ASCII*)

-AND-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *AND* -) [*84*,*84*] *83*)
-ITERATE-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *ITERATE* -) [*84*,*84*] *83*)

translations
-AND-d ⇌ *CONST AND-d*
-ITERATE-d ⇌ *CONST ITERATE-d*

definition *STAR-d* :: (*'a* :: *world*) *monitor* ⇒ *'a formula* ⇒ *'a monitor*
where
STAR-d a f ≡ ((*FIRST LIFT*(◇ *f*)) *ITERATE* (*a*))

definition *REPEAT-d* :: (*'a* :: *world*) *monitor* ⇒ *'a formula* ⇒ *'a monitor*
where
REPEAT-d a w ≡ ((*HALT w*) *ITERATE* (*a WITH LIFT*(*keep*(¬ (*init w*)))))

syntax
-STAR-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *STAR* -) [*84*,*84*] *83*)
-REPEAT-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *REPEATUNTIL* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-STAR-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *STAR* -) [*84*,*84*] *83*)
-REPEAT-d :: [*'a monitor*,*lift*] ⇒ *'a monitor* ((- *REPEATUNTIL* -) [*84*,*84*] *83*)

translations
-STAR-d ⇌ *CONST STAR-d*
-REPEAT-d ⇌ *CONST REPEAT-d*

3.14.3 Monitor Laws

lemma *MFixFst*:
⊢ (ℳ *a*) = ▷ (ℳ *a*)
proof
(*induct a*)
case (*mFIRST-d x*)
then show *?case*
proof −
have *1*: ⊢ (ℳ (*FIRST x*)) = ▷ *x* **by** *simp*
have *2*: ⊢ ▷ *x* = ▷ (▷ *x*) **using** *FstFixFst* **by** *fastforce*
have *3*: ⊢ ▷ (▷ *x*) = ▷(ℳ (*FIRST x*)) **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed
next
case (*mUPTO-d a1 a2*)
then show *?case*
proof −
have *1*: ⊢ (ℳ (*a1 UPTO a2*)) = ▷((ℳ *a1*) ∨ (ℳ *a2*))
by (*simp*)
have *2*: ⊢ ▷((ℳ *a1*) ∨ (ℳ *a2*)) = ▷(▷((ℳ *a1*) ∨ (ℳ *a2*)))
using *FstFixFst* **by** *fastforce*

```
  have 3: ⊢ ▷(▷((M a1) ∨ (M a2))) = ▷(M (a1 UPTO a2))
      using 2 by simp
  from 1 2 3 show ?thesis by fastforce
 qed
next
  case (mTHRU-d a1 a2)
  then show ?case
  proof −
   have 1: ⊢ (M (a1 THRU a2)) = ▷( di (M a1) ∧ di(M a2))
      by (simp)
   have 2: ⊢ ▷( di (M a1) ∧ di(M a2)) = ▷(▷(di(M a1) ∧ di(M a2)))
      using FstFixFst by fastforce
   have 3: ⊢ ▷(▷( di (M a1) ∧ di(M a2))) = ▷(M (a1 THRU a2))
      using 2 by simp
   from 1 2 3 show ?thesis by fastforce
 qed
next
  case (mTHEN-d a1 a2)
  then show ?case
  proof −
   have 1: ⊢ (M (a1 THEN a2)) = (M a1) ⌢ (M a2)
      by (simp)
   have 2: ⊢ (M a1) ⌢ (M a2) = ▷(M a1) ⌢ ▷(M a2)
      using SChopEqvSChop mTHEN-d.hyps(1) mTHEN-d.hyps(2) by blast
   have 3: ⊢ ▷(M a1) ⌢ ▷(M a2) = ▷(▷(M a1) ⌢ (M a2))
      using FstFstChopEqvFstChopFst
      by (metis FstImpFinite Prop10 inteq-reflection schop-d-def)
   have 4: ⊢ ▷(▷(M a1) ⌢ (M a2)) = ▷((M a1) ⌢ (M a2))
      using FstEqvRule LeftSChopEqvSChop mTHEN-d.hyps(1) by (metis inteq-reflection)
   have 5: ⊢ ▷((M a1) ⌢ (M a2)) = ▷(M (a1 THEN a2))
      using 4 by simp
   from 1 2 3 4 5 show ?thesis by fastforce
  qed
next
  case (mWITH-d a x2)
  then show ?case
  proof −
   have 1: ⊢ (M (a WITH x2)) = ((M a) ∧ ( x2))
      by (simp )
   have 2: ⊢ ((M a) ∧ ( x2)) = (▷(M a) ∧ ( x2))
      using mWITH-d.hyps by fastforce
   have 3: ⊢ (▷(M a) ∧ ( x2)) = ▷(▷(M a) ∧ ( x2))
      using FstFstAndEqvFstAnd by fastforce
   have 4: ⊢ ▷(▷(M a) ∧ ( x2)) = ▷((M a) ∧ ( x2))
      using 2 FstEqvRule by fastforce
   have 5: ⊢ ▷((M a) ∧ ( x2)) = ▷(M (a WITH x2))
      using 4 by simp
   from 1 2 3 4 5 show ?thesis by (metis inteq-reflection)
 qed
qed
```

lemma *MGuardFalseEqvFalse*:
$\vdash \ \mathcal{M}(GUARD\ \#False) = \#False$
proof −
have *1*: $\vdash \mathcal{M}(GUARD\ \#False) = \mathcal{M}(EMPTY\ WITH\ LIFT(init\ \#False))$ **by** (*simp add*: *GUARD-d-def*)
have *2*: $\vdash \mathcal{M}(EMPTY\ WITH\ LIFT(init\ \#False)) = (\mathcal{M}(EMPTY) \wedge (init\ \#False))$ **by** (*simp*)
have *3*: $\vdash \#False = (init\ \#False)$ **by** (*simp add*:*init-defs Valid-def*)
have *4*: $\vdash (\mathcal{M}(EMPTY) \wedge (init\ \#False)) = (\mathcal{M}(EMPTY) \wedge \#False)$ **using** *3* **by** *auto*
have *5*: $\vdash (\mathcal{M}(EMPTY) \wedge \#False) = \#False$ **by** *simp*
have *6*: $\vdash (\mathcal{M}(EMPTY) \wedge (init\ \#False)) = \#False$ **using** *4 5* **by** *simp*
have *7*: $\vdash \mathcal{M}(EMPTY\ WITH\ LIFT(init\ \#False)) = \#False$ **using** *2 6* **by** *fastforce*
have *8*: $\vdash \mathcal{M}(GUARD\ \#False) = \#False$ **using** *1 7* **by** *fastforce*
from *8* **show** *?thesis* **by** *auto*
qed

lemma *MFirstFalseEqvFalse*:
$\vdash \ \mathcal{M}(FIRST\ LIFT\ \#False) = \#False$
proof −
have *1*: $\vdash \mathcal{M}(FIRST\ LIFT\ \#False) = \triangleright\ \#False$ **by** (*simp*)
have *2*: $\vdash \mathcal{M}(FIRST\ LIFT\ \#False) = \#False$ **using** *FstFalse* **by** *fastforce*
from *2* **show** *?thesis* **by** *auto*
qed

lemma *MFailAlt*:
$\vdash \ \mathcal{M}\ FAIL = \#False$
proof −
have *1*: $\vdash \ \mathcal{M}\ FAIL = \ \mathcal{M}\ (GUARD\ (\#False))$ **by** (*simp add*: *FAIL-d-def*)
have *2*: $\vdash \mathcal{M}(GUARD\ (\#False)) = \#False$ **using** *MGuardFalseEqvFalse* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailEqvFirstFalseWithinEmpty*:
$FAIL \simeq \ ((FIRST\ LIFT\ \#False)\ WITHIN\ empty)$
proof −
have *1*: $\vdash \mathcal{M}\ (\ (FIRST\ LIFT\ \#False)\ WITHIN\ (\ empty\ )) =$
$\mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)\ )$
by (*simp add*: *WITHIN-d-def*)
have *2*: $\vdash \ \mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)\ ) =$
$(\mathcal{M}(FIRST\ LIFT\ \#False) \wedge (Limit\ empty\ ))$
by (*simp*)
have *3*: $\vdash \ \mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)\ ) = \#False$
using *MFirstFalseEqvFalse* **by** *auto*
have *4*: $\vdash \mathcal{M}(\ (FIRST\ LIFT\ \#False)\ WITHIN\ (\ empty\ )) = \#False$
using *1 3* **by** *fastforce*
have *5*: $\vdash \mathcal{M}(\ FAIL) = \#False$
using *MFailAlt* **by** *simp*
from *4 5* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyAlt*:

$\vdash \mathcal{M}\ EMPTY = empty$
proof −
have *1*: $\vdash \mathcal{M}\ (\ EMPTY) = \mathcal{M}(\ (FIRST\ LIFT\ empty))$ **by** (*simp add*: *EMPTY-d-def*)
have *2*: $\vdash \mathcal{M}\ (\ (FIRST\ LIFT\ empty)) = \rhd\ empty$ **by** (*simp*)
have *3*: $\vdash \rhd\ empty = empty$ **using** *FstEmpty* **by** *auto*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MSkipAlt*:
$\vdash \mathcal{M}\ SKIP = skip$
proof −
have *1*: $\vdash \mathcal{M}\ SKIP = \mathcal{M}\ (FIRST\ LIFT\ skip)$ **by** (*simp add*: *SKIP-d-def*)
have *2*: $\vdash \mathcal{M}\ (FIRST\ LIFT\ skip) = \rhd\ skip$ **by** (*simp*)
have *3*: $\vdash \rhd\ skip = skip$ **using** *FstSkip* **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MGuardAlt*:
$\vdash \mathcal{M}\ (GUARD(w)) = (empty \wedge init\ w)$
proof −
have *1*: $\vdash \mathcal{M}(GUARD(w)) = \mathcal{M}(EMPTY\ WITH\ (\ LIFT\ (init\ w)))$ **by** (*simp add*:*GUARD-d-def*)
have *2*: $\vdash \mathcal{M}(EMPTY\ WITH\ (\ LIFT(init\ w))) = (\mathcal{M}(\ EMPTY) \wedge (\ init\ w))$ **by** (*simp*)
have *3*: $\vdash (\mathcal{M}(\ EMPTY) \wedge (\ init\ w)) = (empty \wedge (\ init\ w))$ **using** *MEmptyAlt* **by** *fastforce*
have *4*: $\vdash\ (empty \wedge (\ init\ w)) = (empty \wedge init\ w)$ **by** *simp*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MLengthAlt*:
$\vdash \mathcal{M}\ (LEN(k)) = len(k)$
proof −
have *1*: $\vdash \mathcal{M}(LEN(k)) = \mathcal{M}(FIRST\ LIFT(len(k)))$ **by** (*simp add*:*LEN-d-def*)
have *2*: $\vdash \mathcal{M}(FIRST\ LIFT(len(k))) = \rhd(len(k))$ **by** (*simp*)
have *3*: $\vdash \rhd(len(k)) = len(k)$ **using** *FstLenEqvLen* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MAlwaysAlt*:
$\vdash \mathcal{M}(a\ ALWAYS\ w) = (\mathcal{M}(a) \wedge \Box\ (init\ w))$
proof −
have *1*: $\vdash \mathcal{M}(a\ ALWAYS\ w) = \mathcal{M}(a\ WITH\ LIFT(bi\ (finite \longrightarrow fin\ (init\ w))))$
by (*simp add*: *ALWAYS-d-def*)
have *2*: $\vdash \mathcal{M}(a\ WITH\ LIFT(bi\ (finite \longrightarrow fin\ (init\ w)))) = (\mathcal{M}(a) \wedge (bi\ (finite \longrightarrow fin\ (init\ w))))$
by (*simp*)
have *3*: $\vdash (\mathcal{M}(a) \wedge (bi\ (finite \longrightarrow fin\ (init\ w)))) = (\mathcal{M}(a) \wedge \Box\ (init\ w))$
using *BoxStateEqvBiFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MSometimeAlt*:

$\vdash \mathcal{M}(a\ SOMETIME\ w) = (\mathcal{M}(a) \wedge \Diamond\ (init\ w))$
proof −
have *1*: $\vdash \mathcal{M}(a\ SOMETIME\ w) = \mathcal{M}(a\ WITH\ LIFT(di\ (finite \wedge fin\ (init\ w))))$
by (*simp add*: *SOMETIME-d-def*)
have *2*: $\vdash \mathcal{M}(a\ WITH\ LIFT(di\ (finite \wedge fin\ (init\ w)))) = (\mathcal{M}(a) \wedge (di\ (finite \wedge fin\ (init\ w))))$
by (*simp*)
have *3*: $\vdash \mathcal{M}(a\ WITH\ LIFT(di\ (finite \wedge fin\ (init\ w)))) = (\mathcal{M}(a) \wedge\ \Diamond\ (init\ w))$
using *DiamondStateEqvDiFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MWithinAlt*:
$\vdash \mathcal{M}(a\ WITHIN\ f) = (\mathcal{M}(a) \wedge (bs\ (\neg\ f)))$
proof −
have *1*: $\vdash \mathcal{M}(a\ WITHIN\ f) = \mathcal{M}(a\ WITH\ LIFT(bs\ (\neg\ f)))$
by (*simp add*: *WITHIN-d-def LIMIT-d-def*)
have *2*: $\vdash \mathcal{M}(a\ WITH\ LIFT(bs\ (\neg\ f))) = (\mathcal{M}(a) \wedge (bs\ (\neg\ f)))$
by (*simp*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MTimesAlt*:
$\vdash \mathcal{M}(a\ TIMES\ k) = fpower\ (\mathcal{M}(a))\ k$
proof
(*induct k*)
case *0*
then show *?case*
proof −
have *1*: $\vdash\ \mathcal{M}\ (a\ TIMES\ 0) =\ \mathcal{M}\ EMPTY$ **by** *simp*
have *2*: $\vdash\ \mathcal{M}\ EMPTY = empty$ **using** *MEmptyAlt* **by** *simp*
have *3*: $\vdash empty = fpower\ (\mathcal{M}\ a)\ 0$ **by** (*simp add*: *fpower-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed
next
case (*Suc k*)
then show *?case*
proof −
have *1*: $\vdash\ \mathcal{M}(\ a\ TIMES\ Suc\ k) = \mathcal{M}(a\ THEN\ (a\ TIMES\ k)\ )$
by *simp*
have *2*: $\vdash \mathcal{M}(a\ THEN\ (a\ TIMES\ k)\ ) = (\mathcal{M}\ a)^\frown(\mathcal{M}\ (a\ TIMES\ k))$
by (*simp*)
have *3*: $\vdash (\mathcal{M}\ a)^\frown(\mathcal{M}(a\ TIMES\ k)) = (\mathcal{M}\ a)^\frown(fpower\ (\mathcal{M}\ a)\ k)$
using *RightSChopEqvSChop Suc.hyps* **by** *blast*
have *4*: $\vdash (\mathcal{M}\ a)^\frown(fpower\ (\mathcal{M}\ a)\ k) = fpower\ (\mathcal{M}\ a)\ (Suc\ k)$
by (*simp add*: *fpower-d-def schop-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *MUptoAlt*:

$\vdash \mathcal{M}(a\ UPTO\ b) =$
$((\ (\mathcal{M}\ a) \wedge bi\ (\neg(\mathcal{M}\ b)) \wedge finite) \vee ((\mathcal{M}\ b) \wedge bi\ (\neg(\mathcal{M}\ a)) \wedge finite) \vee ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b)))$
proof −
have *1*: $\vdash \mathcal{M}\ (a\ UPTO\ b) = \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))$
by (*simp*)
have *2*: $\vdash \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = ((\triangleright(\mathcal{M}\ a) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee (\triangleright(\mathcal{M}\ b) \wedge\ (bs\ (\neg(\mathcal{M}\ a)))))$
using *FstWithOrEqv* **by** *blast*
have *3*: $\vdash ((\triangleright(\mathcal{M}\ a) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee (\triangleright(\mathcal{M}\ b) \wedge\ (bs\ (\neg(\mathcal{M}\ a))))) =$
$(((\mathcal{M}\ a) \wedge ((\mathcal{M}\ b) \vee \neg(\mathcal{M}\ b)) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee$
$((\mathcal{M}\ b) \wedge ((\mathcal{M}\ a) \vee \neg(\mathcal{M}\ a)) \wedge\ (bs\ (\neg(\mathcal{M}\ a)))))$
using *MFixFst* **by** *fastforce*
have *4*: $\vdash (((\mathcal{M}\ a) \wedge ((\mathcal{M}\ b) \vee \neg(\mathcal{M}\ b)) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee$
$((\mathcal{M}\ b) \wedge ((\mathcal{M}\ a) \vee \neg(\mathcal{M}\ a)) \wedge\ (bs\ (\neg(\mathcal{M}\ a))))) =$
$(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b)))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ ))$
by *auto*
have *5*: $\vdash (((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b)))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \wedge (\ (\triangleright(\mathcal{M}\ b)) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b)))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ (\triangleright(\mathcal{M}\ a)) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ ))$
by (*simp add*: *first-d-def*)
have *6*: $\vdash (((\mathcal{M}\ a) \wedge (\ (\triangleright(\mathcal{M}\ b)) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b)))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ (\triangleright(\mathcal{M}\ a)) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b)))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ ))$
using *MFixFst* **by** *fastforce*
have *7*: $\vdash (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))) = (bi(\neg(\mathcal{M}\ b)) \wedge finite)$
using *AndBsEqvBi* **by** *blast*
have *8*: $\vdash (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) = (bi(\neg(\mathcal{M}\ a)) \wedge finite)$
using *AndBsEqvBi* **by** *blast*
have *9*: $\vdash (((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee ((\neg(\mathcal{M}\ b)) \wedge bs(\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee ((\neg(\mathcal{M}\ a)) \wedge bs(\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\ bi(\neg(\mathcal{M}\ b)) \wedge finite\ \ ))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\ bi(\neg(\mathcal{M}\ a)) \wedge finite))\ ))$
using *7 8* **by** *fastforce*
have *10*: $\vdash(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\ bi(\neg(\mathcal{M}\ b)) \wedge finite))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\ bi(\neg(\mathcal{M}\ a)) \wedge finite))\ )) =$
$(((\ (\mathcal{M}\ a) \wedge (\mathcal{M}\ b)) \vee (\ (\mathcal{M}\ a) \wedge bi(\neg(\mathcal{M}\ b)) \wedge finite)) \vee$
$((\ (\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) \vee (\ (\mathcal{M}\ b) \wedge bi(\neg(\mathcal{M}\ a)) \wedge finite)))$
by *auto*
have *11*: $\vdash (((\ (\mathcal{M}\ a) \wedge (\mathcal{M}\ b)) \vee (\ (\mathcal{M}\ a) \wedge bi(\neg(\mathcal{M}\ b)) \wedge finite)) \vee$
$((\ (\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) \vee (\ (\mathcal{M}\ b) \wedge bi(\neg(\mathcal{M}\ a)) \wedge finite))) =$
$((\ (\mathcal{M}\ a) \wedge bi\ (\neg(\mathcal{M}\ b)) \wedge finite) \vee ((\mathcal{M}\ b) \wedge bi\ (\neg(\mathcal{M}\ a)) \wedge finite) \vee ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b)))$
by *auto*
show *?thesis*
by (*metis 10 11 2 3 4 5 6 9 MON.simps(2) int-eq*)
qed

lemma *MThruAlt*:
$\vdash \mathcal{M}(a\ THRU\ b) = (((\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee ((\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$

proof −
  have *1*: ⊢ $\mathcal{M}(a\ THRU\ b) = \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$
      by (*simp*)
  have *2*: ⊢ $\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = ((\triangleright(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (\triangleright(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$
      using *FstDiAndDiEqv* **by** *auto*
  have *3*: ⊢ $((\triangleright(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (\triangleright(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))) =$
          $(((\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee ((\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$
      using *MFixFst* **by** *fastforce*
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MHaltAlt*:
⊢ $\mathcal{M}(HALT\ w) = (halt(init\ w) \wedge finite)$
proof −
  have *1*: ⊢ $\mathcal{M}(HALT\ w) = \mathcal{M}(FIRST\ LIFT(fin\ (init\ \ w)))$ **by** (*simp add*: *HALT-d-def*)
  have *2*: ⊢ $\mathcal{M}(FIRST\ LIFT(fin\ (init\ \ w))) = \triangleright\ (fin\ (init\ \ w))$ **by** (*simp*)
  have *3*: ⊢ $\triangleright\ (fin\ (init\ \ w)) = (halt(init\ w) \wedge finite)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
  from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailUpto*:
$(FAIL\ UPTO\ a) \simeq (\ a)$
proof −
  have *1*: ⊢ $\mathcal{M}(FAIL\ UPTO\ a) = \triangleright(\ (\mathcal{M}\ FAIL) \vee (\mathcal{M}\ a))$ **by** (*simp*)
  have *2*: ⊢ $(\mathcal{M}\ FAIL \vee \mathcal{M}\ a) = (\#False \vee \mathcal{M}\ a)$ **using** *MFailAlt* **by** *auto*
  have *3*: ⊢ $\triangleright(\mathcal{M}\ FAIL \vee (\mathcal{M}\ a)) = \triangleright(\#False \vee (\mathcal{M}\ a))$ **using** *2 FstEqvRule* **by** *blast*
  have *4*: ⊢ $(\#False \vee (\mathcal{M}\ a)) = \mathcal{M}\ a$ **by** *simp*
  have *5*: ⊢ $\triangleright(\#False \vee (\mathcal{M}\ a)) = \ \triangleright(\mathcal{M}\ a)$ **using** *4 FstEqvRule* **by** *blast*
  have *6*: ⊢ $\triangleright(\mathcal{M}\ a) = \mathcal{M}\ a$ **using** *MFixFst* **by** *fastforce*
  from *1 2 3 4 5 6* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailThru*:
  $(FAIL\ THRU\ (\ a)) \simeq \ \ FAIL$
proof −
  have *1*: ⊢ $\mathcal{M}\ (FAIL\ THRU\ (\ a)) = \triangleright(di(\mathcal{M}\ FAIL) \wedge di(\mathcal{M}\ a))$
      by (*simp*)
  have *2*: ⊢ $\triangleright(di\ (\mathcal{M}\ FAIL) \wedge di(\mathcal{M}\ a)) = \triangleright(di\ (\#False) \wedge di(\mathcal{M}\ a))$
      using *MFailAlt* **by** (*metis 1 int-eq*)
  have *3*: ⊢ $di\ \#False = \#False$
      by (*simp add*: *di-defs Valid-def*)
  hence *4*: ⊢ $\triangleright(di\ (\#False) \wedge di(\mathcal{M}\ a)) = \triangleright(\ (\#False) \wedge di(\mathcal{M}\ a))$
      by (*metis 2 inteq-reflection*)
  have *5*: ⊢ $\triangleright(\ (\#False) \wedge di(\mathcal{M}\ a)) = \triangleright\#False$
      using *FstEqvRule* **by** *fastforce*
  have *6*: ⊢ $\triangleright\#False = \#False$ **using** *FstFalse*
      by *auto*
  have *7*: ⊢ $\#False = \mathcal{M}\ FAIL$
      using *MFailAlt* **by** *auto*
 from *1 2 4 5 6 7* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MFailAnd*:
(*FAIL AND a*) ≃ *FAIL*
proof −
have *1*: ⊢ $\mathcal{M}$ (*FAIL AND a*) = ($\mathcal{M}$ *FAIL* ∧ ($\mathcal{M}$ *a*)) **by** (*simp add*: *AND-d-def*)
have *2*: ⊢ ($\mathcal{M}$ *FAIL* ∧ ($\mathcal{M}$ *a*)) = (#*False* ∧ ($\mathcal{M}$ *a*)) **using** *MFailAlt* **by** *fastforce*
have *3*: ⊢ (#*False* ∧ ($\mathcal{M}$ *a*)) = #*False* **by** *auto*
have *4*: ⊢ $\mathcal{M}$(*FAIL AND a*) = #*False* **using** *1 2 3* **by** *fastforce*
have *5*: ⊢ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4 5* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenFail*:
(*a THEN FAIL*) ≃ *FAIL*
proof −
have *1*: ⊢ $\mathcal{M}$ (*a THEN FAIL*) = ($\mathcal{M}$ *a*)⌢($\mathcal{M}$ *FAIL*) **by** (*simp*)
have *2*: ⊢ ($\mathcal{M}$ *a*)⌢($\mathcal{M}$ *FAIL*) = ($\mathcal{M}$ *a*)⌢#*False* **by** (*simp add*: *MFailAlt RightSChopEqvSChop*)
have *3*: ⊢ ($\mathcal{M}$ *a*)⌢#*False* = #*False***by** (*simp add*: *SChopRightFalse*)
have *4*: ⊢ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailThen*:
(*FAIL THEN a*) ≃ *FAIL*
proof −
have *1*: ⊢ $\mathcal{M}$(*FAIL THEN a*) = ($\mathcal{M}$ *FAIL*)⌢($\mathcal{M}$ *a*) **by** (*simp*)
have *2*: ⊢ ($\mathcal{M}$ *FAIL*)⌢($\mathcal{M}$ *a*) = #*False*⌢($\mathcal{M}$ *a*) **using** *MFailAlt* **using** *LeftSChopEqvSChop* **by** *blast*
have *3*: ⊢ #*False*⌢($\mathcal{M}$ *a*) = #*False* **by** (*simp add*: *schop-d-def chop-d-def Valid-def*)
have *4*: ⊢ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailWith*:
(*FAIL WITH f*) ≃ *FAIL*
proof −
have *1*: ⊢ $\mathcal{M}$ (*FAIL WITH f*) = (($\mathcal{M}$ *FAIL*) ∧ *f*) **by** (*simp*)
have *2*: ⊢ (($\mathcal{M}$ *FAIL*) ∧ *f*) = (#*False* ∧ *f*) **using** *MFailAlt* **by** *auto*
have *3*: ⊢ (#*False* ∧ *f*) = #*False* **by** *simp*
have *4*: ⊢ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithFalse*:
(*a WITH* (*LIFT*(#*False*))) ≃ *FAIL*
proof −
have *1*: ⊢ $\mathcal{M}$ (*a WITH LIFT*(#*False*)) = (($\mathcal{M}$ *a*) ∧ #*False*) **by** (*simp*)
have *2*: ⊢ (($\mathcal{M}$ *a*) ∧ #*False*) = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithTrue*:
(*a WITH* (*LIFT*(#*True*))) ≃ *a*
proof −
have *1*: ⊢ $\mathcal{M}$ (*a WITH LIFT*(#*True*)) = (($\mathcal{M}$ *a*) ∧ #*True*) **by** (*simp*)
have *2*: ⊢ (($\mathcal{M}$ *a*) ∧ #*True*) = $\mathcal{M}$ *a* **by** *simp*
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyUpto*:
(*EMPTY UPTO a*) ≃ *EMPTY*
proof −
have *1*: ⊢ $\mathcal{M}$ (*EMPTY UPTO a*) = ▷($\mathcal{M}$ *EMPTY* ∨ ($\mathcal{M}$ *a*)) **by** (*simp*)
have *2*: ⊢ $\mathcal{M}$ *EMPTY* = *empty* **using** *MEmptyAlt* **by** *auto*
hence *3*: ⊢ ($\mathcal{M}$ *EMPTY* ∨ ($\mathcal{M}$ *a*)) = (*empty* ∨ ($\mathcal{M}$ *a*)) **by** *auto*
hence *4*: ⊢ ▷($\mathcal{M}$ *EMPTY* ∨ $\mathcal{M}$ *a*) = ▷(*empty* ∨ $\mathcal{M}$ *a*) **using** *FstEqvRule* **by** *blast*
have *5*: ⊢ ▷(*empty* ∨ $\mathcal{M}$ *a*) = *empty* **using** *FstEmptyOrEqvEmpty* **by** *blast*
have *6*: ⊢ *empty* = $\mathcal{M}$ *EMPTY* **using** *MEmptyAlt* **by** *auto*
from *1 4 5 6* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyThru*:
(*EMPTY THRU a*) ≃ (*a*)
proof −
have *1*: ⊢ $\mathcal{M}$(*EMPTY THRU a*) = ▷(*di*($\mathcal{M}$ *EMPTY*) ∧ *di*($\mathcal{M}$ *a*)) **by** (*simp*)
have *2*: ⊢ *di*($\mathcal{M}$ *EMPTY*) = *di empty* **using** *MEmptyAlt DiEqvDi* **by** *blast*
hence *3*: ⊢ (*di*($\mathcal{M}$ *EMPTY*) ∧ *di*($\mathcal{M}$ *a*)) = (*di empty* ∧ *di*($\mathcal{M}$ *a*)) **by** *auto*
hence *4*: ⊢ (*di empty* ∧ *di*($\mathcal{M}$ *a*)) = *di*($\mathcal{M}$ *a*) **using** *DiEmpty* **by** *auto*
have *5*: ⊢ (*di*($\mathcal{M}$ *EMPTY*) ∧ *di*($\mathcal{M}$ *a*)) = *di*($\mathcal{M}$ *a*) **using** *3 4* **by** *fastforce*
hence *6*: ⊢ ▷(*di*($\mathcal{M}$ *EMPTY*) ∧ *di*($\mathcal{M}$ *a*)) = ▷(*di*($\mathcal{M}$ *a*)) **using** *FstEqvRule* **by** *blast*
have *7*: ⊢ ▷(*di*($\mathcal{M}$ *a*)) = ▷($\mathcal{M}$ *a*) **using** *FstDiEqvFst* **by** *blast*
have *8*: ⊢ ▷($\mathcal{M}$ *a*) = ($\mathcal{M}$ *a*) **using** *MFixFst* **by** *fastforce*
from *1 6 7 8* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenEmpty*:
(*a THEN EMPTY*) ≃ (*a WITH* (*LIFT finite*))
proof −
have *1*: ⊢ $\mathcal{M}$(*a THEN EMPTY*) = ($\mathcal{M}$ *a*)⌢ ($\mathcal{M}$ *EMPTY*) **by** (*simp*)
have *2*: ⊢ ($\mathcal{M}$ *a*)⌢ ($\mathcal{M}$ *EMPTY*) = ($\mathcal{M}$ *a*)⌢ *empty* **by** (*simp add*: *MEmptyAlt RightSChopEqvSChop*)
have *3*: ⊢ ($\mathcal{M}$ *a*)⌢ *empty* = (($\mathcal{M}$ *a*) ∧ *finite*) **by** (*simp add*: *ChopEmpty schop-d-def*)
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis MON.simps*(*5*) *inteq-reflection*)
qed

lemma *MEmptyThen*:
(*EMPTY THEN a*) ≃ *a*
proof −
have *1*: ⊢ $\mathcal{M}$(*EMPTY THEN a*) = ($\mathcal{M}$ *EMPTY*)⌢($\mathcal{M}$ *a*) **by** (*simp*)
have *2*: ⊢ ($\mathcal{M}$ *EMPTY*)⌢($\mathcal{M}$ *a*) = *empty*⌢($\mathcal{M}$ *a*) **by** (*simp add*: *MEmptyAlt LeftSChopEqvSChop*)
have *3*: ⊢ *empty*⌢($\mathcal{M}$ *a*) = ($\mathcal{M}$ *a*) **by** (*simp add*: *EmptySChop*)

from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyIterate*:
(*EMPTY ITERATE b*) ≃ *EMPTY*
proof −
have *1*: ⊢ $\mathcal{M}$(*EMPTY ITERATE b*) = $\mathcal{M}$(*EMPTY WITH LIFT* (*schopstar* ($\mathcal{M}$ *b*)))
by (*simp add*: *ITERATE-d-def*)
have *2*: ⊢ $\mathcal{M}$ (*EMPTY WITH LIFT* (*schopstar* ($\mathcal{M}$ *b*))) = ($\mathcal{M}$ *EMPTY* ∧ (*schopstar* ($\mathcal{M}$ *b*)))
by (*simp*)
have *3*: ⊢ ($\mathcal{M}$ *EMPTY* ∧ (*schopstar* ($\mathcal{M}$ *b*))) = (*empty* ∧ (*schopstar* ($\mathcal{M}$ *b*)))
using *MEmptyAlt* **by** *auto*
have *4*: ⊢ (*empty* ∧ (*schopstar* ($\mathcal{M}$ *b*))) = (*empty* ∧ (*empty* ∨ ((($\mathcal{M}$ *b*) ∧ *more*)⌢(*schopstar* ($\mathcal{M}$ *b*)))))
using *SChopstarEqv* **by** *fastforce*
have *5*: ⊢ (*empty* ∧ (*empty* ∨ ((($\mathcal{M}$ *b*) ∧ *more*)⌢(*schopstar* ($\mathcal{M}$ *b*))))) = *empty*
by *auto*
have *6*: ⊢ $\mathcal{M}$(*EMPTY ITERATE b*) = $\mathcal{M}$ *EMPTY*
using *1 2 3 4 5 MEmptyAlt* **by** *fastforce*
from *6* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MIterateIdemp*:
(*a ITERATE a*) ≃ (*a*)
proof −
have *1*: ⊢ $\mathcal{M}$(*a ITERATE a*) = $\mathcal{M}$ (*a WITH LIFT* (*schopstar* ($\mathcal{M}$ *a*))) **by** (*simp add*: *ITERATE-d-def*)
have *2*: ⊢ $\mathcal{M}$(*a WITH LIFT* (*schopstar* ($\mathcal{M}$ *a*))) = (($\mathcal{M}$ *a*) ∧ (*schopstar* ($\mathcal{M}$ *a*))) **by** (*simp*)
have *3*: ⊢ (($\mathcal{M}$ *a*) ∧ (*schopstar* ($\mathcal{M}$ *a*))) = (▷($\mathcal{M}$ *a*) ∧ (*schopstar* (▷($\mathcal{M}$ *a*)))) **using** *MFixFst*
by (*metis 2 inteq-reflection*)
have *4*: ⊢ (▷($\mathcal{M}$ *a*) ∧ (*schopstar* (▷($\mathcal{M}$ *a*)))) = ▷($\mathcal{M}$ *a*) **using** *FstAndFstStarEqvFst* **by** *fastforce*
have *5*: ⊢ ▷($\mathcal{M}$ *a*) = $\mathcal{M}$ *a* **using** *MFixFst* **by** *fastforce*
from *1 2 3 4 5* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoIdemp*:
(*a UPTO a*) ≃ (*a*)
proof −
have *1*: ⊢ $\mathcal{M}$(*a UPTO a*) = ▷(($\mathcal{M}$ *a*) ∨ ($\mathcal{M}$ *a*)) **by** *auto*
have *2*: ⊢ ▷(($\mathcal{M}$ *a*) ∨ ($\mathcal{M}$ *a*)) = ▷($\mathcal{M}$ *a*) **using** *FstEqvRule* **by** *fastforce*
have *3*: ⊢ ▷($\mathcal{M}$ *a*) = ($\mathcal{M}$ *a*) **using** *MFixFst* **by** *fastforce*
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruIdemp*:
(*a THRU a*) ≃ (*a*)
proof −
have *1*: ⊢ $\mathcal{M}$(*a THRU a*) = ▷(*di*($\mathcal{M}$ *a*) ∧ *di*($\mathcal{M}$ *a*)) **by** *auto*
have *2*: ⊢ ▷(*di*($\mathcal{M}$ *a*) ∧ *di*($\mathcal{M}$ *a*)) = ▷(*di* ($\mathcal{M}$ *a*)) **using** *FstEqvRule* **by** *fastforce*
have *3*: ⊢ ▷(*di* ($\mathcal{M}$ *a*)) = ▷($\mathcal{M}$ *a*) **using** *FstDiEqvFst* **by** *blast*
have *4*: ⊢ ▷($\mathcal{M}$ *a*) = ($\mathcal{M}$ *a*) **using** *MFixFst* **by** *fastforce*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MAndIdemp*:
$(a\ AND\ a) \simeq (a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ AND\ a) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a))$ **by** (*simp add*: *AND-d-def*)
have *2*: $\vdash ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) = (\mathcal{M}\ a)$ **by** *fastforce*
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithIdemp*:
$(\ (a\ WITH\ f)\ \ WITH\ f) \simeq (a\ WITH\ f)$
proof −
have *1*: $\vdash \mathcal{M}(\ (a\ WITH\ f)\ \ WITH\ f) = (((\mathcal{M}\ a) \wedge (\ f)) \wedge (\ f))$ **by** *auto*
have *2*: $\vdash (((\mathcal{M}\ a) \wedge (\ f)) \wedge (\ f)) = ((\mathcal{M}\ a) \wedge (\ f))$ **by** *fastforce*
have *3*: $\vdash ((\mathcal{M}\ a) \wedge (\ f)) = \mathcal{M}(a\ WITH\ f)$ **by** *auto*
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoCommut*:
$(a\ UPTO\ b) \simeq (b\ UPTO\ a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ UPTO\ b) = \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))$ **by** (*simp*)
have *2*: $\vdash ((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = ((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **by** *auto*
hence *3*: $\vdash \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *blast*
have *4*: $\vdash \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ a)) = \mathcal{M}(b\ UPTO\ a)$ **by** *auto*
from *1 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruCommut*:
$(a\ THRU\ b) \simeq (b\ THRU\ a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ THRU\ b) = \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$ **by** (*simp*)
have *2*: $\vdash (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))$ **by** *auto*
hence *3*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *blast*
have *4*: $\vdash \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)) = \mathcal{M}(b\ THRU\ a)$ **by** *auto*
from *1 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MAndCommut*:
$(a\ AND\ b) \simeq (b\ AND\ a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ AND\ b) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b))$ **by** (*simp add*: *AND-d-def*)
have *2*: $\vdash ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b)) = ((\mathcal{M}\ b) \wedge (\mathcal{M}\ a))$ **by** *auto*
have *3*: $\vdash ((\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) = \mathcal{M}(b\ AND\ a)$ **by** (*simp add*: *AND-d-def*)
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithCommut*:
$((a\ WITH\ f)\ \ WITH\ g) \simeq ((a\ WITH\ g)\ \ WITH\ f)$

proof −
  have *1*: $\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ g) = (((\mathcal{M}\ a) \wedge (f)) \wedge (g))$ **by** *auto*
  have *2*: $\vdash (((\mathcal{M}\ a) \wedge (f)) \wedge (g)) = (((\mathcal{M}\ a) \wedge (g)) \wedge (f))$ **by** *auto*
  have *3*: $\vdash (((\mathcal{M}\ a) \wedge (g)) \wedge (f)) = \mathcal{M}((a\ WITH\ g)\ WITH\ f)$ **by** *auto*
 from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithAbsorp*:
 $((a\ WITH\ f)\ WITH\ g) \simeq (a\ WITH\ LIFT(f \wedge g))$
proof −
  have *1*: $\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ g) = (((\mathcal{M}\ a) \wedge (f)) \wedge (g))$ **by** *auto*
  have *2*: $\vdash (((\mathcal{M}\ a) \wedge (f)) \wedge (g)) = ((\mathcal{M}\ a) \wedge (f \wedge g))$ **by** *auto*
 from *1 2* **show** *?thesis* **by** (*simp add*: *MonEq*)
qed

lemma *MUptoAssoc*:
 $((a\ UPTO\ b)\ UPTO\ c) \simeq (a\ UPTO\ (b\ UPTO\ c))$
proof −
  have *1*: $\vdash \mathcal{M}((a\ UPTO\ b)\ UPTO\ c) = \triangleright(\mathcal{M}(a\ UPTO\ b) \vee (\mathcal{M}\ c))$
      by (*simp*)
  have *2*: $\vdash \triangleright(\mathcal{M}(a\ UPTO\ b) \vee (\mathcal{M}\ c)) = \triangleright(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c))$
      by *auto*
  have *3*: $\vdash \triangleright(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = \triangleright(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c))$
      using *FstFstOrEqvFstOrL* **by** *blast*
  have *4*: $\vdash (((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = ((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
      by *auto*
  hence *5*: $\vdash \triangleright(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = \triangleright((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
      using *FstEqvRule* **by** *blast*
  have *6*: $\vdash \triangleright((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \triangleright((\mathcal{M}\ a) \vee \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
      using *FstFstOrEqvFstOrR* **by** *fastforce*
  have *7*: $\vdash \triangleright((\mathcal{M}\ a) \vee \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \triangleright((\mathcal{M}\ a) \vee \mathcal{M}(b\ UPTO\ c))$
      by *auto*
  have *8*: $\vdash \triangleright((\mathcal{M}\ a) \vee \mathcal{M}(b\ UPTO\ c)) = \mathcal{M}(a\ UPTO\ (b\ UPTO\ c))$
      by *auto*
 from *1 2 3 5 6 7 8* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *DiAndFiniteEqvDiFst*:
$\vdash$ *di* $(f \wedge$ *finite*$) =$ *di*$(\triangleright f)$
proof −
 have *1*: $\vdash (\triangleright f \wedge$ *finite*$) = \triangleright f$
   by (*meson FstImpFinite Prop10 Prop11*)
 have *2*: $\vdash (f \wedge$ *finite*$)$;#*True* $= (\triangleright f \wedge$ *finite*$)$;#*True*
   using *DfEqvDfFst FstImpFinite* **unfolding** *df-d-def schop-d-def* **by** *auto*
 show *?thesis*
 by (*metis 1 2 di-d-def inteq-reflection*)
qed

lemma *FstEqvFstAndFinite*:
$\vdash \triangleright f = \triangleright (f \wedge$ *finite*$)$

by (*metis FstDfEqvFst FstDiEqvFst di-d-def inteq-reflection itl-def*(*15*) *schop-d-def*)

lemma *DiAndFiniteEqvDfAndFinite*:
$\vdash$ (*di f* $\wedge$ *finite*) = (*df f* $\wedge$ *finite*)
by (*auto simp add*: *Valid-def di-defs df-defs finite-defs*)

lemma *FstDiAndDiEqvFstDfAndDf*:
$\vdash$ $\rhd$(*di f* $\wedge$ *di g*) = $\rhd$(*df f* $\wedge$ *df g*)
proof −
have *1*: $\vdash$ $\rhd$(*di f* $\wedge$ *di g*) = $\rhd$ ((*di f* $\wedge$ *di g*) $\wedge$ *finite*)
by (*simp add*: *FstEqvFstAndFinite*)
have *2*: $\vdash$ ((*di f* $\wedge$ *di g*) $\wedge$ *finite*) = ((*df f* $\wedge$ *df g*) $\wedge$ *finite*)
using *DiAndFiniteEqvDfAndFinite*[*of f*] *DiAndFiniteEqvDfAndFinite*[*of g*]
by *fastforce*
have *3*: $\vdash$ $\rhd$((*di f* $\wedge$ *di g*) $\wedge$ *finite*) = $\rhd$((*df f* $\wedge$ *df g*) $\wedge$ *finite*)
by (*simp add*: *2 FstEqvRule*)
have *4*: $\vdash$ $\rhd$((*df f* $\wedge$ *df g*) $\wedge$ *finite*) = $\rhd$(*df f* $\wedge$ *df g*)
by (*meson FstEqvFstAndFinite Prop11*)
show *?thesis*
by (*metis 1 3 4 inteq-reflection*)
qed

lemma *MThruAssoc*:
((*a THRU b*) *THRU c*) $\simeq$ (*a THRU* (*b THRU c*))
proof −
have *1*: $\vdash$ $\mathcal{M}$((*a THRU b*) *THRU c*) = $\rhd$(*di*($\rhd$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*))) $\wedge$ *di*($\mathcal{M}$ *c*))
by *auto*
have *2*: $\vdash$ *di*($\rhd$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*))) = *di*((*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*)) $\wedge$ *finite*)
using *DiAndFiniteEqvDiFst* **by** *fastforce*
have *3*: $\vdash$ *di*((*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*)) $\wedge$ *finite*) = (*di*(($\mathcal{M}$ *a*) $\wedge$ *finite*) $\wedge$ *di*(($\mathcal{M}$ *b*) $\wedge$ *finite*))
by (*metis DfDfAndEqvDf DiAndFiniteEqvDiFst MFixFst df-d-def di-d-def inteq-reflection schop-d-def*)
have *4*: $\vdash$ *di*($\rhd$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*))) = (*di*(($\mathcal{M}$ *a*) $\wedge$ *finite*) $\wedge$ *di*(($\mathcal{M}$ *b*) $\wedge$ *finite*))
using *2 3* **by** *fastforce*
hence *5*: $\vdash$ (*di*($\rhd$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*))) $\wedge$ *di*(($\mathcal{M}$ *c*) $\wedge$ *finite*)) =
(*di*(($\mathcal{M}$ *a*) $\wedge$ *finite*) $\wedge$ (*di*(($\mathcal{M}$ *b*) $\wedge$ *finite*) $\wedge$ *di*(($\mathcal{M}$ *c*) $\wedge$ *finite*)))
by *auto*
have *6*: $\vdash$ (*di*(($\mathcal{M}$ *b*) $\wedge$ *finite*) $\wedge$ *di*(($\mathcal{M}$ *c*) $\wedge$ *finite*)) = *di* ((*di*($\mathcal{M}$ *b*) $\wedge$ *di*($\mathcal{M}$ *c*)) $\wedge$ *finite*)
by (*metis DfDfAndEqvDf DfEqvDiAndFinite DiAndFiniteEqvDiFst MFixFst inteq-reflection*)
have *7*: $\vdash$ *di* ((*di*($\mathcal{M}$ *b*) $\wedge$ *di*($\mathcal{M}$ *c*)) $\wedge$ *finite*) = *di* ($\rhd$(*di*($\mathcal{M}$ *b*) $\wedge$ *di*($\mathcal{M}$ *c*)))
by (*simp add*: *DiAndFiniteEqvDiFst*)
have *8*: $\vdash$ (*di*(($\mathcal{M}$ *b*) $\wedge$ *finite*) $\wedge$ *di*(($\mathcal{M}$ *c*) $\wedge$ *finite*)) =
di ($\rhd$(*di*($\mathcal{M}$ *b*) $\wedge$ *di*($\mathcal{M}$ *c*)))
using *6 7* **by** *fastforce*
hence *9*: $\vdash$(*di*(($\mathcal{M}$ *a*) $\wedge$ *finite*) $\wedge$ (*di*(($\mathcal{M}$ *b*) $\wedge$ *finite*) $\wedge$ *di*(($\mathcal{M}$ *c*) $\wedge$ *finite*))) =
(*di*(($\mathcal{M}$ *a*) $\wedge$ *finite*) $\wedge$ *di* ($\rhd$(*di*($\mathcal{M}$ *b*) $\wedge$ *di*($\mathcal{M}$ *c*))))
by *auto*
have *10*: $\vdash$ (*di*($\rhd$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*))) $\wedge$ *di*(($\mathcal{M}$ *c*) $\wedge$ *finite*)) =
(*di*(($\mathcal{M}$ *a*) $\wedge$ *finite*) $\wedge$ *di* ($\rhd$(*di*($\mathcal{M}$ *b*) $\wedge$ *di*($\mathcal{M}$ *c*))))

using *5 9* **by** *fastforce*
hence *11*: $\vdash \rhd(di(\rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) \ \wedge di((\mathcal{M}\ c) \wedge finite)) =$
$\rhd(di((\mathcal{M}\ a) \wedge finite) \wedge di\ (\rhd(di(\mathcal{M}\ b) \ \wedge di(\mathcal{M}\ c))))$
using *FstEqvRule* **by** *fastforce*
have *12*: $\vdash \rhd(di((\mathcal{M}\ a) \wedge finite) \wedge di\ (\rhd(di(\mathcal{M}\ b) \ \wedge di(\mathcal{M}\ c)))) = \mathcal{M}(a\ THRU\ (b\ THRU\ c))$
by (*metis DiAndFiniteEqvDiFst MFixFst MON.simps*(*3*) *inteq-reflection*)
from *1 11 12* **show** *?thesis* **using** *MonEq*
by (*metis DiAndFiniteEqvDiFst MFixFst inteq-reflection*)
qed

lemma *MAndAssoc*:
$((a\ AND\ b)\ AND\ c) \simeq (a\ AND\ (b\ AND\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ AND\ b)\ AND\ c) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b) \wedge (\mathcal{M}\ c))$
using *AND-d-def* **by** (*metis MON.simps*(*5*) *MWithAbsorp eq-d-def*)
have *2*: $\vdash ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b) \wedge (\mathcal{M}\ c)) = \mathcal{M}\ (a\ AND\ (b\ AND\ c))$
using *AND-d-def* **by** (*simp add*: *AND-d-def*)
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenAssoc*:
$((a\ THEN\ b)\ THEN\ c) \simeq (a\ THEN\ (b\ THEN\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ THEN\ b)\ THEN\ c) = ((\mathcal{M}\ a)\frown(\mathcal{M}\ b))\frown(\mathcal{M}\ c)$ **by** *auto*
have *2*: $\vdash ((\mathcal{M}\ a)\frown(\mathcal{M}\ b))\frown(\mathcal{M}\ c) = (\mathcal{M}\ a)\frown((\mathcal{M}\ b)\frown(\mathcal{M}\ c))$ **by** (*meson Prop11 SChopAssoc*)
have *3*: $\vdash (\mathcal{M}\ a)\frown((\mathcal{M}\ b)\frown(\mathcal{M}\ c)) = \mathcal{M}(a\ THEN\ (b\ THEN\ c))$ **by** *auto*
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoThruAbsorp*:
$(a\ UPTO\ (a\ THRU\ b)) \simeq \ a$
proof −
have *1*: $\vdash \mathcal{M}(a\ UPTO\ (a\ THRU\ b)) = \rhd((\mathcal{M}\ a) \vee \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ ))$
by *simp*
have *2*: $\vdash \rhd((\mathcal{M}\ a) \vee \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$\rhd((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ ))$
using *FstFstOrEqvFstOrR* **by** *auto*
have *3*: $\vdash \ ((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$(((\mathcal{M}\ a) \vee di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
by *auto*
have *4*: $\vdash (((\mathcal{M}\ a) \vee di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
using *OrDiEqvDi* **by** *fastforce*
have *5*: $\vdash ((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
using *3 4* **by** *auto*
hence *6*: $\vdash \rhd\ ((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$\rhd\ ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
using *FstEqvRule* **by** *blast*
have *7*: $\vdash \rhd\ ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) =$

$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg(\ (di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))))$
by (*auto simp add*: *first-d-def*)
have *8*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) =$
$((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
by *auto*
hence *9*: $\vdash (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(\neg((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *fastforce*
have *10*: $\vdash (\neg((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(\neg(((\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *AndDiEqv* **using** *5* **by** *auto*
have *11*: $\vdash (\neg(((\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
by *auto*
have *12*: $\vdash (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
using *9 10 11* **by** *auto*
hence *13*: $\vdash bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$bs\ (\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
using *BsEqvRule* **by** *blast*
have *14*: $\vdash bs\ ((\neg(\mathcal{M}\ a)) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) =$
$(bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *BsAndEqv* **by** *fastforce*
have *141*: $\vdash bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *13 14* **by** *fastforce*
hence *15*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))))) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *auto*
have *16*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((bs\ ((\neg(\mathcal{M}\ a))) \wedge di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *auto*
have *17*: $\vdash ((bs\ ((\neg(\mathcal{M}\ a))) \wedge di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((\triangleright(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *FstEqvBsNotAndDi* **by** *fastforce*
have *18*: $\vdash ((\triangleright(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(((\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *MFixFst* **by** *fastforce*
have *19*: $\vdash (((\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(((\mathcal{M}\ a)) \wedge\ \ bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$

by *auto*
have *20*: $\vdash \quad (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = (\neg(di(\mathcal{M}\ a)) \vee \neg(di(\mathcal{M}\ b))\ )$
by *auto*
have *21*: $\vdash (\neg(di(\mathcal{M}\ a)) \vee \neg(di(\mathcal{M}\ b))\ ) = ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi\ (\neg(\mathcal{M}\ b)))\ )$
by (*simp add*: *bi-d-def*)
have *22*: $\vdash (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi\ (\neg(\mathcal{M}\ b))))$
using *20 21* **by** *auto*
hence *23*: $\vdash bs\ (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = bs\ ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi(\neg(\mathcal{M}\ b))))$
using *BsEqvRule* **by** *blast*
have *24*: $\vdash bs\ ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi\ (\neg(\mathcal{M}\ b)))) = bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))$
using *BsOrBsEqvBsBiOrBi* **by** *fastforce*
have *25*: $\vdash bs\ (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))$
using *23 24* **using** *BsOrBsEqvBsBiOrBi* **by** *fastforce*
hence *26*: $\vdash ((\mathcal{M}\ a) \wedge bs\ (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
by *auto*
have *27*: $\vdash ((\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$(\triangleright(\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
using *MFixFst* **by** *fastforce*
have *28*: $\vdash (\triangleright(\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
by (*auto simp add*: *first-d-def*)
have *29*: $\vdash ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)))$
by *auto*
have *30*: $\vdash ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) = \triangleright(\mathcal{M}\ a)$
by (*simp add*: *first-d-def*)
have *31*: $\vdash \triangleright(\mathcal{M}\ a) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
have *32*: $\vdash \mathcal{M}(a\ UPTO\ (a\ THRU\ b)) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg(\ (di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))))$
using *1 2 6 7* **by** *fastforce*
have *33*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg(\ (di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))))) =$
$(((\mathcal{M}\ a)) \wedge\ \ bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *15 16 17 18 19* **by** (*metis int-eq*)
have *34*: $\vdash (((\mathcal{M}\ a)) \wedge\ \ bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) = (\mathcal{M}\ a)$
using *26 27 28 29 30 31* **by** (*metis int-eq*)
from *32 33 34* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruUptoAbsorp*:
$(a\ THRU\ (a\ UPTO\ b)) \simeq (a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ THRU\ (a\ UPTO\ b)) = \triangleright(di(\mathcal{M}\ a) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))))$
by *simp*
have *2*: $\vdash\ \ \triangleright(di(\mathcal{M}\ a) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)))) =$
$\triangleright(di(\mathcal{M}\ a) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b))))$
by (*metis DfEqvDfFst FstDiAndDiEqvFstDfAndDf inteq-reflection*)

have *3*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)))) =$
$\triangleright(di(\mathcal{M}\ a) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
by (*metis DiOrEqv FstEqvRule inteq-reflection lift-and-com*)
have *4*: $\vdash (di(\mathcal{M}\ a) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) = (di(\mathcal{M}\ a))$
by *auto*
hence *5*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) = \triangleright(di(\mathcal{M}\ a))$
using *FstEqvRule* **by** *blast*
have *6*: $\vdash \triangleright(di(\mathcal{M}\ a)) = \triangleright(\mathcal{M}\ a)$
using *FstDiEqvFst* **by** *blast*
have *7*: $\vdash \triangleright(\mathcal{M}\ a) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
from *1 2 3 5 6 7* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoThruDistrib*:
$(a\ UPTO\ (b\ THRU\ c)) \simeq ((a\ UPTO\ b)\ THRU\ (a\ UPTO\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ UPTO\ b)\ THRU\ (a\ UPTO\ c)) =$
$\triangleright(\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ )$
by *simp*
have *2*: $\vdash (\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$(\ di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \wedge finite) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ c)) \wedge finite)\ )$
using *DiAndFiniteEqvDiFst* **by** *fastforce*
have *3*: $\vdash (\ di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \wedge finite) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ c)) \wedge finite)\ ) =$
$((\ di((\mathcal{M}\ a) \wedge finite) \vee di((\mathcal{M}\ b) \wedge finite)) \wedge (di((\mathcal{M}\ a) \wedge finite) \vee di((\mathcal{M}\ c) \wedge finite)))$
by (*metis* (*no-types*, *lifting*) *DiDiAndEqvDi DiOrEqv FiniteOr inteq-reflection*)
have *4*: $\vdash ((\ di((\mathcal{M}\ a) \wedge finite) \vee di((\mathcal{M}\ b) \wedge finite)) \wedge (di((\mathcal{M}\ a) \wedge finite) \vee di((\mathcal{M}\ c) \wedge finite))) =$
$(di((\mathcal{M}\ a) \wedge finite) \vee (di((\mathcal{M}\ b) \wedge finite) \wedge di((\mathcal{M}\ c) \wedge finite)))$
by *auto*
have *5*: $\vdash (\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$(di((\mathcal{M}\ a) \wedge finite) \vee (di((\mathcal{M}\ b) \wedge finite) \wedge di((\mathcal{M}\ c) \wedge finite)))$
using *2 3 4* **by** *fastforce*
hence *6*: $\vdash \triangleright(\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$\triangleright(\ di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ )$
by (*metis DiAndFiniteEqvDiFst MFixFst int-simps*(*1*) *inteq-reflection*)
have *7*: $\vdash \triangleright(\ di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ ) =$
$\triangleright(\ \triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ )$
using *FstFstOrEqvFstOr* **by** *fastforce*
have *8*: $\vdash \triangleright(di(\mathcal{M}\ a)) = \triangleright((\mathcal{M}\ a))$
using *FstDiEqvFst* **by** *blast*
have *9*: $\vdash \triangleright((\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
have *10*: $\vdash \triangleright(di(\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *8 9* **by** *fastforce*
hence *11*: $\vdash (\triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
by *auto*
hence *12*: $\vdash \triangleright(\triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$\triangleright((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
using *FstEqvRule* **by** *blast*

have *13*: $\vdash \triangleright((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) = \mathcal{M}(a\ UPTO\ (b\ THRU\ c))$
by *simp*
from *1 6 7 12 13* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruUptoDistrib*:
$(a\ THRU\ (b\ UPTO\ c)) \simeq ((a\ THRU\ b)\ UPTO\ (a\ THRU\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ THRU\ b)\ UPTO\ (a\ THRU\ c)) =$
$\triangleright(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c)))$
by *simp*
have *2*: $\vdash \triangleright(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c))) =$
$\triangleright((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c)))$
using *FstFstOrEqvFstOr* **by** *auto*
have *3*: $\vdash ((di((\mathcal{M}\ a) \wedge finite) \wedge di((\mathcal{M}\ b) \wedge finite)) \vee (di((\mathcal{M}\ a) \wedge finite) \wedge di((\mathcal{M}\ c) \wedge finite))) =$
$(di((\mathcal{M}\ a) \wedge finite) \wedge (di((\mathcal{M}\ b) \wedge finite) \vee di((\mathcal{M}\ c) \wedge finite)))$ **by** *auto*
have *4*: $\vdash (di((\mathcal{M}\ a) \wedge finite) \wedge (di((\mathcal{M}\ b) \wedge finite) \vee di((\mathcal{M}\ c) \wedge finite))) =$
$(di((\mathcal{M}\ a) \wedge finite) \wedge di(\ ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) \wedge finite\ )\ )$ **using** *DiOrEqv*
by (*metis 3 FiniteOr inteq-reflection*)
have *5*: $\vdash (di((\mathcal{M}\ a) \wedge finite) \wedge di(\ ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) \wedge finite)) =$
$(di((\mathcal{M}\ a) \wedge finite) \wedge di(\triangleright(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
using *DiAndFiniteEqvDiFst* **by** *fastforce*
have *6*: $\vdash ((di((\mathcal{M}\ a) \wedge finite) \wedge di((\mathcal{M}\ b) \wedge finite)) \vee (di((\mathcal{M}\ a) \wedge finite) \wedge di((\mathcal{M}\ c) \wedge finite))) =$
$(di((\mathcal{M}\ a) \wedge finite) \wedge di(\triangleright(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
using *3 4 5* **by** *fastforce*
hence *7*: $\vdash \triangleright((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c))) =$
$\triangleright(di(\mathcal{M}\ a) \wedge di(\triangleright(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
by (*metis DiAndFiniteEqvDiFst MFixFst int-simps*(*20*) *inteq-reflection*)
have *8*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di(\triangleright(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c)))) =$
$\mathcal{M}(a\ THRU\ (b\ UPTO\ c))$ **by** *simp*
from *1 2 7 8* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruUptoRDistrib*:
$((a\ THRU\ b)\ UPTO\ c) \simeq ((a\ UPTO\ c)\ THRU\ (b\ UPTO\ c))$
proof −
have *1*: $((a\ THRU\ b)\ UPTO\ c) \simeq (c\ UPTO\ (a\ THRU\ b))$
using *MUptoCommut* **by** *auto*
have *2*: $(c\ UPTO\ (a\ THRU\ b)) \simeq ((c\ UPTO\ a)\ THRU\ (c\ UPTO\ b))$
using *MUptoThruDistrib* **by** *auto*
have *3*: $(c\ UPTO\ a) \simeq (a\ UPTO\ c)$
using *MUptoCommut* **by** *auto*
have *4*: $(c\ UPTO\ b) \simeq (b\ UPTO\ c)$
using *MUptoCommut* **by** *auto*
have *5*: $((c\ UPTO\ a)\ THRU\ (c\ UPTO\ b)) \simeq ((a\ UPTO\ c)\ THRU\ (c\ UPTO\ b))$
using *3* **by** (*simp add*: *MonEqRefl MonEqSubstThru*)
have *6*: $((a\ UPTO\ c)\ THRU\ (c\ UPTO\ b)) \simeq ((c\ UPTO\ b)\ THRU\ (a\ UPTO\ c))$
using *MThruCommut* **by** *auto*
have *7*: $((c\ UPTO\ b)\ THRU\ (a\ UPTO\ c)) \simeq ((b\ UPTO\ c)\ THRU\ (a\ UPTO\ c))$
using *4* **by** (*simp add*: *MonEqRefl MonEqSubstThru*)

from *1 2 5 6 7* **show** *?thesis* **using** *MThruCommut MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoThruRDistrib*:
$((a\ UPTO\ b)\ THRU\ c) \simeq ((a\ THRU\ c)\ UPTO\ (b\ THRU\ c))$
proof −
have *1*: $((a\ UPTO\ b)\ THRU\ c) \simeq (c\ THRU\ (a\ UPTO\ b))$
using *MThruCommut* **by** *auto*
have *2*: $(c\ THRU\ (a\ UPTO\ b)) \simeq ((c\ THRU\ a)\ UPTO\ (c\ THRU\ b))$
using *MThruUptoDistrib* **by** *auto*
have *3*: $(c\ THRU\ a) \simeq (a\ THRU\ c)$
using *MThruCommut* **by** *auto*
have *4*: $(c\ THRU\ b) \simeq (b\ THRU\ c)$
using *MThruCommut* **by** *auto*
have *5*: $((c\ THRU\ a)\ UPTO\ (c\ THRU\ b)) \simeq ((a\ THRU\ c)\ UPTO\ (c\ THRU\ b))$
using *3* **by** (*simp add*: *MonEqRefl MonEqSubstUpto*)
have *6*: $((a\ THRU\ c)\ UPTO\ (c\ THRU\ b)) \simeq ((c\ THRU\ b)\ UPTO\ (a\ THRU\ c))$
using *MUptoCommut* **by** *auto*
have *7*: $((c\ THRU\ b)\ UPTO\ (a\ THRU\ c)) \simeq ((b\ THRU\ c)\ UPTO\ (a\ THRU\ c))$
using *4* **by** (*simp add*: *MonEqRefl MonEqSubstUpto*)
from *1 2 5 6 7* **show** *?thesis* **using** *MUptoCommut MonEq* **by** (*metis int-eq*)
qed

lemma *MWithAndDistrib*:
$((a\ AND\ b)\ WITH\ f) \simeq ((a\ WITH\ f)\ AND\ (b\ WITH\ f))$
proof −
have *1*: $\vdash \mathcal{M}((a\ AND\ b)\ WITH\ f) = (\mathcal{M}(a\ AND\ b) \wedge f)$
by (*simp*)
have *2*: $\vdash \mathcal{M}(a\ AND\ b) = \mathcal{M}(a\ WITH\ LIFT(\mathcal{M}\ b))$
by (*simp add*: *AND-d-def*)
have *3*: $\vdash (\mathcal{M}(a\ AND\ b) \wedge f) = (\mathcal{M}(a\ WITH\ LIFT(\mathcal{M}\ b)) \wedge f)$
using *2* **by** *auto*
have *4*: $\vdash \mathcal{M}(a\ WITH\ (LIFT(\ (\mathcal{M}\ b) \wedge f))) = (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f)$
by *simp*
have *5*: $\vdash (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f) = ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f))$
by *auto*
have *6*: $\vdash ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f)) = (\mathcal{M}(a\ WITH\ f) \wedge \mathcal{M}(\ b\ WITH\ f))$
by *simp*
have *7*: $\vdash (\mathcal{M}(a\ WITH\ f) \wedge \mathcal{M}(b\ WITH\ f)) = \mathcal{M}((a\ WITH\ f)\ WITH\ LIFT(\mathcal{M}(b\ WITH\ f)))$
by *simp*
have *8*: $\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ LIFT(\mathcal{M}(b\ WITH\ f))) = \mathcal{M}((a\ WITH\ f)\ AND\ (b\ WITH\ f))$
by (*simp add*: *AND-d-def*)
from *1 2 3 4 5 6 7 8* **show** *?thesis* **using** *MonEq* **by** (*metis AND-d-def MWithAbsorp int-eq*)
qed

lemma *MHaltWithAndDistrib*:
$(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) \simeq ((HALT\ w)\ WITH\ LIFT(f \wedge g))$
proof −
have *1*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) =$
$\mathcal{M}(((HALT\ w)\ WITH\ f)\ WITH\ LIFT(\mathcal{M}((HALT\ w)\ WITH\ g)))$

by (*simp add*: *AND-d-def*)
have *2*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ WITH\ LIFT(\mathcal{M}((HALT\ w)\ WITH\ g))) =$
$(\mathcal{M}(HALT\ w) \wedge f \wedge \mathcal{M}(HALT\ w) \wedge g)$
by *auto*
have *3*: $\vdash (\mathcal{M}(HALT\ w) \wedge f \wedge \mathcal{M}(HALT\ w) \wedge g) = (\mathcal{M}(HALT\ w) \wedge f \wedge g)$
by *auto*
have *4*: $\vdash (\mathcal{M}(HALT\ w) \wedge f \wedge g) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
by *auto*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MHaltWithUptoHaltWithEqvHaltWithOr*:
$(((HALT\ w)\ WITH\ f)\ UPTO\ ((HALT\ w)\ WITH\ g)) \simeq ((HALT\ w)\ WITH\ LIFT(f \vee g))$
proof −
have *1*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ UPTO\ ((HALT\ w)\ WITH\ g)) =$
$\triangleright(\mathcal{M}((HALT\ w)\ WITH\ f) \vee \mathcal{M}((HALT\ w)\ WITH\ g))$
by (*simp*)
have *2*: $\vdash \triangleright(\mathcal{M}((HALT\ w)\ WITH\ f) \vee \mathcal{M}((HALT\ w)\ WITH\ g)) =$
$\triangleright((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g))$
by *auto*
have *3*: $\vdash ((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g)) = (\mathcal{M}(HALT\ w) \wedge (f \vee g))$
by *auto*
have *4*: $\vdash \triangleright((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g)) = \triangleright(\mathcal{M}(HALT\ w) \wedge (f \vee g))$
using *3 FstEqvRule* **by** *blast*
have *5*: $\vdash \triangleright(\mathcal{M}(HALT\ w) \wedge (f \vee g)) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \vee g)))$
by *simp*
have *6*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ LIFT(f \vee g))) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \vee g)))$
using *MFixFst* **by** *blast*
from *1 2 3 4 5 6* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MHaltWithThruHaltWithEqvHaltWithAndHaltWith*:
$(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) \simeq (((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g))$
proof −
have *1*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) =$
$\triangleright(\ di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)\ )$
by *simp*
have *2*: $\vdash (di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) =$
$(di((halt(init\ w) \wedge finite) \wedge f) \wedge di((halt(init\ w) \wedge finite) \wedge g))$
using *MHaltAlt* **by** (*metis DiDiAndEqvDi inteq-reflection*)
have *3*: $\vdash (di((halt(init\ w)\ \wedge finite) \wedge f) \wedge di((halt(init\ w) \wedge finite) \wedge g)) =$
$di(\ (halt(init\ w) \wedge finite) \wedge f \wedge g)$
by (*metis FstDiamondStateEqvHalt LFstAndDistrD inteq-reflection*)
have *4*: $\vdash di(\ (halt(init\ w) \wedge finite) \wedge f \wedge g) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
by (*metis 3 MHaltAlt inteq-reflection*)
have *5*: $\vdash (di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using *2 3 4* **by** *fastforce*
have *6*: $\vdash \triangleright(di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = \triangleright(di(\mathcal{M}(HALT\ w) \wedge f \wedge g))$
using *5 FstEqvRule* **by** *blast*

have *7*: $\vdash \rhd(di(\mathcal{M}(HALT\ w) \wedge f \wedge g)) = \rhd(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using *FstDiEqvFst* **by** *fastforce*
have *8*: $\vdash \rhd(\mathcal{M}(HALT\ w) \wedge f \wedge g) = \rhd(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
by *simp*
have *9*: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \rhd(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
using *MFixFst* **by** *blast*
have *10*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using *1 2 3 4 5 6 7 8 9 int-eq* **by** *metis*
have *11*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using *MHaltWithAndDistrib* **using** *eq-d-def* **by** *blast*
have *12*: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g))$
using *11* **by** *fastforce*
from *10 12* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenAndDistrib*:
$(a\ THEN\ (b\ AND\ c)) \simeq ((a\ THEN\ b)\ AND\ (a\ THEN\ c))$
proof −
have *1*: $\vdash \mathcal{M}(a\ THEN\ (b\ AND\ c)) = (\mathcal{M}(a)) \frown (\mathcal{M}(b\ AND\ c))$
by *simp*
have *2*: $\vdash (\mathcal{M}(a)) \frown (\mathcal{M}(b\ AND\ c)) = (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c))$
by (*simp add*: *AND-d-def*)
have *3*: $\vdash (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c)) = \rhd (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c))$
by (*simp add*: *LeftSChopEqvSChop MFixFst*)
have *4*: $\vdash \rhd (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c)) = ((\rhd (\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge (\rhd (\mathcal{M}(a)) \frown (\mathcal{M}(c))))$
by (*metis LFstAndDistrB Prop11 schop-d-def*)
have *5*: $\vdash((\rhd (\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge (\rhd (\mathcal{M}(a)) \frown (\mathcal{M}(c)))) =$
$((\ (\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge (\ (\mathcal{M}(a)) \frown (\mathcal{M}(c)))\ )$ **using** *MFixFst*
by (*metis 4 inteq-reflection*)
have *6*: $\vdash ((\ (\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge (\ (\mathcal{M}(a)) \frown (\mathcal{M}(c)))\ ) =$
$(\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c))$
by *simp*
have *7*: $\vdash (\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c)) = \mathcal{M}((a\ THEN\ b)\ AND\ (a\ THEN\ c))$
by (*simp add*: *AND-d-def*)
from *1 2 3 4 5 6 7* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenUptoDistrib*:
$(a\ THEN\ (b\ UPTO\ c)) \simeq ((a\ THEN\ b)\ UPTO\ (a\ THEN\ c))$
proof −
have *1*: $\vdash (\mathcal{M}\ (a\ THEN\ (b\ UPTO\ c))) = ((\mathcal{M}\ a) \frown (\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
by *simp*
have *2*: $\vdash ((\mathcal{M}\ a) \frown (\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))) = (\rhd(\mathcal{M}\ a) \frown (\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
by (*simp add*: *MFixFst LeftSChopEqvSChop*)
have *3*: $\vdash (\rhd(\mathcal{M}\ a) \frown (\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))) = ((\rhd(\rhd(\mathcal{M}\ a) \frown ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))))$
by (*metis FstFstChopEqvFstChopFst FstImpFinite Prop10 inteq-reflection schop-d-def*)
have *4*: $\vdash \rhd(\mathcal{M}\ a) \frown ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) = (\mathcal{M}\ a) \frown ((\mathcal{M}\ b) \vee (\mathcal{M}\ c))$
using *MFixFst* **by** (*metis LeftSChopEqvSChop inteq-reflection*)
have *5*: $\vdash (\mathcal{M}\ a) \frown ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) = ((\mathcal{M}\ a) \frown (\mathcal{M}\ b) \vee (\mathcal{M}\ a) \frown (\mathcal{M}\ c))$

by (*simp add*: *SChopOrEqv*)
have *6*: $\vdash ((\mathcal{M}\ a)\frown(\mathcal{M}\ b) \vee (\mathcal{M}\ a)\frown(\mathcal{M}\ c)) = (\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c))$
by *simp*
have *7*: $\vdash \rhd(\mathcal{M}\ a)\frown((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) = (\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c))$
using *6 5 4* **by** *fastforce*
have *8*: $\vdash \rhd(\rhd(\mathcal{M}\ a)\frown((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \rhd(\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c))$
using *7* **by** (*simp add*: *FstEqvRule*)
have *9*: $\vdash \rhd(\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c)) = \mathcal{M}((a\ THEN\ b)\ UPTO\ (a\ THEN\ c))$
by *simp*
from *9 7 1 2 3* **show** *?thesis* **by** (*metis eq-d-def inteq-reflection*)
qed

lemma *MThenthruDistrib*:
$(a\ THEN\ (b\ THRU\ c)) \simeq ((a\ THEN\ b)\ THRU\ (a\ THEN\ c))$
proof −
have *1*: $\vdash \mathcal{M}(a\ THEN\ (b\ THRU\ c)) = (\mathcal{M}\ a)\frown\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))$
by *simp*
have *2*: $\vdash (\mathcal{M}\ a)\frown\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)) = \rhd(\mathcal{M}\ a)\frown\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))$
by (*simp add*: *MFixFst LeftSChopEqvSChop*)
have *3*: $\vdash \rhd(\mathcal{M}\ a)\frown\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)) = \rhd(\rhd(\mathcal{M}\ a)\frown(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)\ ))$
by (*metis FstFstChopEqvFstChopFst FstImpFinite Prop10 inteq-reflection schop-d-def*)
have *4*: $\vdash \rhd(\mathcal{M}\ a)\frown(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)\ ) = (\rhd(\mathcal{M}\ a)\frown di(\mathcal{M}\ b) \wedge \rhd(\mathcal{M}\ a)\frown di(\mathcal{M}\ c))$
by (*metis LFstAndDistrB inteq-reflection schop-d-def*)
have *5*: $\vdash (\rhd(\mathcal{M}\ a)\frown di(\mathcal{M}\ b) \wedge \rhd(\mathcal{M}\ a)\frown di(\mathcal{M}\ c)) = ((\mathcal{M}\ a)\frown di(\mathcal{M}\ b) \wedge (\mathcal{M}\ a)\frown di(\mathcal{M}\ c))$
using *MFixFst* **by** (*metis 4 int-eq*)
have *6*: $\vdash (\mathcal{M}\ a)\frown di(\mathcal{M}\ b) = (\mathcal{M}\ a)\frown((\mathcal{M}\ b)\frown\#True)$
by (*metis DiAndFiniteEqvDiFst MFixFst RightSChopEqvSChop di-d-def inteq-reflection schop-d-def*)
have *7*: $\vdash (\mathcal{M}\ a)\frown((\mathcal{M}\ b)\frown\#True) = ((\mathcal{M}\ a)\frown(\mathcal{M}\ b))\frown\#True$
by (*simp add*: *SChopAssoc*)
have *8*: $\vdash ((\mathcal{M}\ a)\frown(\mathcal{M}\ b))\frown\#True = di((\mathcal{M}\ a)\frown(\mathcal{M}\ b))$
by (*metis DiAndFiniteEqvDiFst MFixFst MON.simps(4) di-d-def inteq-reflection schop-d-def*)
have *9*: $\vdash (\mathcal{M}\ a)\frown di(\mathcal{M}\ b) = di((\mathcal{M}\ a)\frown(\mathcal{M}\ b))$
using *8 7 6* **by** *fastforce*
have *10*: $\vdash (\mathcal{M}\ a)\frown di(\mathcal{M}\ c) = (\mathcal{M}\ a)\frown((\mathcal{M}\ c)\frown\#True)$
by (*metis DiAndFiniteEqvDiFst MFixFst di-d-def int-simps(20) inteq-reflection schop-d-def*)
have *11*: $\vdash (\mathcal{M}\ a)\frown((\mathcal{M}\ c)\frown\#True) = ((\mathcal{M}\ a)\frown(\mathcal{M}\ c))\frown\#True$
by (*simp add*: *SChopAssoc*)
have *12*: $\vdash ((\mathcal{M}\ a)\frown(\mathcal{M}\ c))\frown\#True = di((\mathcal{M}\ a)\frown(\mathcal{M}\ c))$
by (*metis* (*no-types*, *lifting*) *10 11 ChopAssoc di-d-def inteq-reflection schop-d-def*)
have *13*: $\vdash (\mathcal{M}\ a)\frown di(\mathcal{M}\ c) = di((\mathcal{M}\ a)\frown(\mathcal{M}\ c))$
using *12 11 10* **by** *fastforce*
have *14*: $\vdash ((\mathcal{M}\ a)\frown di(\mathcal{M}\ b) \wedge (\mathcal{M}\ a)\frown di(\mathcal{M}\ c)) = (di((\mathcal{M}\ a)\frown(\mathcal{M}\ b)) \wedge di((\mathcal{M}\ a)\frown(\mathcal{M}\ c)))$
using *13 9* **by** *fastforce*
have *15*: $\vdash (di((\mathcal{M}\ a)\frown(\mathcal{M}\ b)) \wedge di((\mathcal{M}\ a)\frown(\mathcal{M}\ c))) = (di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c)))$
by *simp*
have *16*: $\vdash \rhd(\mathcal{M}\ a)\frown(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)\ ) = (di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c)))$
using *15 14 4 5* **by** *fastforce*
have *17*: $\vdash \rhd(\rhd(\mathcal{M}\ a)\frown(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)\ )) = \rhd(di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c)))$
using *16* **by** (*simp add*: *FstEqvRule*)
have *18*: $\vdash \rhd(di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c))) = \mathcal{M}((a\ THEN\ b)\ THRU\ (a\ THEN\ c))$

by *simp*
from *18 16 1 2 3* **show** *?thesis* **by** (*metis eq-d-def int-eq*)
qed

end

3.15 Monitor Example

theory *MonitorExample*
imports
FOTheorems Monitor
begin

locale *Test* =
fixes *v* :: *state* ⇒ *nat*
fixes *y* :: *state* ⇒ *bool*
fixes *z* :: *state* ⇒ *nat*
fixes *F2* :: *nat statefun* ⇒ *temporal*
fixes *F3* :: *bool statefun* ⇒ *temporal*
fixes *F4* :: *nat statefun* ⇒ *temporal*
fixes *F5* :: *nat statefun* ⇒ *temporal*
fixes *Init2* :: *nat statefun* ⇒ *temporal*
fixes *Init3* :: *bool statefun* ⇒ *temporal*
fixes *Mon1* :: *state monitor*
fixes *Mon2* :: *state monitor*
fixes *Mon3* :: *state monitor*
fixes *Mon4* :: *state monitor*
fixes *Mon5* :: *state monitor*
fixes *Mon6* :: *state monitor*
defines *F2* ≡ (λ *v*. *TEMP* □ (*#0* ≤ $*v*))
defines *F3* ≡ (λ *p*. *TEMP* □ ($*p* ∨ ¬ $*p*))
defines *F4* ≡ (λ *z*. *TEMP* $*z*=*#0* ∧ *z gets* $*z*+*#1*)
defines *F5* ≡ (λ *z*. *TEMP fin*($*z*=*#4*))
defines *Init2* ≡ (λ *v*. *TEMP* $*v* = *#0*)
defines *Init3* ≡ (λ *p*. *TEMP* $*p*)
defines *Mon1* ≡ *FIRST*(*F2 v*)
defines *Mon2* ≡ *EMPTY UPTO Mon1*
defines *Mon3* ≡ *Mon1 WITH* (*F2 v*)
defines *Mon4* ≡ *Mon2 THEN Mon1*
defines *Mon5* ≡ *Mon3 THRU Mon4*
defines *Mon6* ≡ (*FIRST F4 z*) *WITH* (*F5 z*)

lemma (**in** *Test*) *test*:
⊢ $\mathcal{M}$(*Mon1*) = *empty*
proof −
have *1*: ⊢ $\mathcal{M}$(*Mon1*) = ▷(□ (*#0* ≤ $*v*))

using *F2-def Mon1-def* **by** *fastforce*
have *2*: $\vdash \Box\ (\ \#0 \leq \$v\ )$
by (*simp add*: *Valid-def itl-defs*)
have *3*: $\vdash \rhd(\Box\ (\ \#0 \leq \$v\ )) = empty$
using *2* **by** (*metis FstTrue int-eq int-eq-true*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test1*:
$\vdash \mathcal{M}(Mon2) = empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon2) = \mathcal{M}(EMPTY\ UPTO\ Mon1)$
using *Mon2-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(EMPTY\ UPTO\ Mon1) = \rhd(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1))$
by *fastforce*
have *3*: $\vdash \rhd(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1)) = \rhd(empty \vee empty)$
using *test* **by** (*metis 2 MEmptyAlt int-eq*)
have *4*: $\vdash \rhd(empty \vee empty) = empty$
using *FstEmptyOrEqvEmpty* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test2*:
$\vdash \mathcal{M}(Mon3) = empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon3) = \mathcal{M}(Mon1\ WITH\ (F2\ v))$ **using** *Mon3-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(Mon1\ WITH\ (F2\ v)) = (\mathcal{M}(Mon1) \wedge (F2\ v))$ **by** *fastforce*
have *3*: $\vdash (\mathcal{M}(Mon1) \wedge (F2\ v)) = (empty \wedge (F2\ v))$ **using** *test* **by** *fastforce*
have *4*: $\vdash (F2\ v)$ **by** (*simp add*: *F2-def Valid-def itl-defs*)
have *5*: $\vdash (empty \wedge (F2\ v)) = empty$ **using** *4* **by** *fastforce*
from *1 2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test3*:
$\vdash \mathcal{M}(Mon4) = empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon4) = \mathcal{M}(Mon2\ THEN\ Mon1)$
using *Mon4-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(Mon2\ THEN\ Mon1) = (\ \mathcal{M}(Mon2)\ ) \frown (\ \mathcal{M}(Mon1)\ )$
by *fastforce*
have *3*: $\vdash (\ \mathcal{M}(Mon2)\ ) \frown (\ \mathcal{M}(Mon1)\ ) = empty \frown empty$
using *test test1* **using** *SChopEqvSChop* **by** *blast*
have *4*: $\vdash empty \frown empty = empty$
by (*simp add*: *EmptySChop*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test4*:
$\vdash \mathcal{M}(Mon5) = empty$
proof −

have *1*: $\vdash \mathcal{M}(Mon5) = \mathcal{M}(Mon3\ THRU\ Mon4)$
using *Mon5-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(Mon3\ THRU\ Mon4) = \triangleright(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4)))$
by *fastforce*
have *3*: $\vdash (di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4))) = (di(empty) \wedge di(empty))$
using *test3 test2* **by** (*metis inteq-reflection lift-and-com*)
hence *4*: $\vdash \triangleright(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4))) = \triangleright(di(empty) \wedge di(empty))$
by (*simp add*: *FstEqvRule*)
have *5*: $\vdash \triangleright(di(empty) \wedge di(empty)) = \triangleright(di(empty))$
by *simp*
have *6*: $\vdash \triangleright(di(empty)) = empty$
using *FstDiEqvFst FstEmpty* **by** *fastforce*
from *6 5 4 2 1* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test5*:
$\vdash \mathcal{M}\ (Mon6) = (\triangleright(\$z{=}\#0\ \wedge\ z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ )$
proof −
have *1*: $\vdash \mathcal{M}(Mon6) = (\mathcal{M}(FIRST\ F4\ z) \wedge (F5\ z))$
using *Mon6-def* **by** *fastforce*
have *2*: $\vdash (\mathcal{M}(FIRST\ F4\ z) \wedge (F5\ z)) = (\triangleright(F4\ z) \wedge fin(\$z{=}\#4))$
using *F5-def* **by** *fastforce*
have *3*: $\vdash (\triangleright(F4\ z) \wedge fin(\$z{=}\#4)) = (\triangleright(\$z{=}\#0 \wedge\ \ z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ )$
using *F4-def* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test5-1*:
$\vdash \triangleright(\$z{=}\#0 \wedge\ \ z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ \longrightarrow$
$\triangleright((\$z{=}\#0 \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4))$

using *FstWithAndImp* **by** *blast*

lemma (**in** *Test*) *test5-2*:
$(s \models (\$z{=}\#0 \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4) \wedge finite) =$
$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0\ \wedge (\forall\ i{<}\ nlength\ s.\ z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge$
$z\ (nnth\ s\ (the\text{-}enat\ (nlength\ s))) = 4)$
apply (*simp add*: *itl-defs nsubn-def1*)
by (*metis ndropn-nfirst nfirst-eq-nnth-zero ntaken-nfirst*)

lemma (**in** *Test*) *test5-3*:
$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0\ \wedge (\forall\ i{<}\ nlength\ s.\ z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge$
$z\ (nnth\ s\ (the\text{-}enat\ (nlength\ s))) = 4)$
$\Longrightarrow$
$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0\ \wedge (\forall\ i{\leq}\ nlength\ s.\ z\ (nnth\ s\ i) = i)$
$\wedge\ z\ (nnth\ s\ (the\text{-}enat\ (nlength\ s))) = 4)$

proof −
assume *a0*: $nfinite\ s \wedge (z\ (nnth\ s\ 0) = 0\ \wedge (\forall\ i{<}\ nlength\ s.\ z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge$

```
   z (nnth s (the-enat (nlength s))) = 4)
  show nfinite s ∧ (z (nnth s 0) =0  ∧ (∀ i≤ nlength s. z (nnth s i) = i)
  ∧ z (nnth s (the-enat (nlength s))) = 4)
  proof −
    have 0: nfinite s using a0 by auto
    have 1: z (nnth s 0) =0  using a0 by auto
    have 2: z (nnth s (the-enat (nlength s))) = 4 using a0 by auto
    have 3: (∀ i≤ nlength s. z (nnth s i) = i)
    proof
      fix i
      show i ≤ nlength s ⟶ z ( nnth s i) = i
      proof
       (induct i)
       case 0
       then show ?case by (simp add: 1)
       next
       case (Suc i)
       then show ?case by (simp add: Suc-ile-eq a0)
       qed
    qed
    from 0 1 2 3 show ?thesis by auto
   qed
qed

lemma (in Test) test5-4:
   (nfinite s ∧ z (nnth s 0) =0  ∧ (∀ i≤ nlength s. z (nnth s i) = i)
  ∧ z (nnth s (the-enat (nlength s))) = 4) ⟹
    (nfinite s ∧ z (nnth s 0) =0  ∧ (∀ i< nlength s. z (nnth s (Suc i)) = Suc(z (nnth s i))) ∧
   z (nnth s (the-enat (nlength s))) = 4)

proof −
 assume a0: (nfinite s ∧ z (nnth s 0) =0  ∧ (∀ i≤ nlength s. z (nnth s i) = i)
  ∧ z (nnth s (the-enat (nlength s))) = 4)
 show (nfinite s ∧ z (nnth s 0) =0  ∧ (∀ i< nlength s. z (nnth s (Suc i)) = Suc(z (nnth s i))) ∧
   z (nnth s (the-enat (nlength s))) = 4)
 proof −
  have 0: nfinite s using a0 by auto
  have 1: z (nnth s 0) =0  using a0 by auto
  have 2: z (nnth s (the-enat (nlength s))) = 4 using a0 by auto
  have 3: (∀ i< nlength s. z (nnth s (Suc i)) = Suc(z (nnth s i))) by (simp add: Suc-ile-eq a0)
  from 0 1 2 3 show ?thesis by auto
 qed
qed

lemma (in Test) test5-5:
  (nfinite s ∧ z (nnth s 0) =0  ∧ (∀ i< nlength s. z (nnth s (Suc i)) = Suc(z (nnth s i))) ∧
   z (nnth s (the-enat (nlength s))) = 4)
  =
 (nfinite s ∧ z (nnth s 0) =0  ∧ (∀ i≤ nlength s. z (nnth s i) = i)
  ∧ z (nnth s (the-enat (nlength s))) = 4)
```

using *test5-3 test5-4* **by** *blast*

lemma (**in** *Test*) *test5-6* :
(*nfinite s* ∧ *z* (*nnth s 0*) =*0* ∧ (∀ *i*≤ *nlength s*. *z* (*nnth s i*) = *i*)
∧ *z* (*nnth s* (*the-enat* (*nlength s*))) = *4*) =
(*nlength s* =*4* ∧ (∀ *i*≤ *nlength s*. *z* (*nnth s i*) = *i*))

by (*metis dual-order.refl nfinite-conv-nlength-enat numeral-eq-enat the-enat.simps zero-enat-def zero-le-numeral*)

lemma (**in** *Test*) *test5-7* :
(*s* ⊨ ($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*) ∧ *finite*) =
(*nlength s* =*4* ∧ (∀ *i*≤ *nlength s*. *z* (*nnth s i*) = *i*))
using *test5-6*[*of s*] *test5-5*[*of s*] *test5-2*[*of s*] **by** *presburger*

lemma (**in** *Test*) *test5-8-0*:
(*s* ⊨ ▷(($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*))) =
(*s* ⊨ ▷((($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*)) ∧ *finite*))
using *FstEqvFstAndFinite*[*of TEMP* ($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*)]
unfolding *Valid-def* **by** *auto*

lemma (**in** *Test*) *test5-8-1*:
(⊢ ▷((($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*)) ∧ *finite*)) =
(⊢ ▷(($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*) ∧ *finite*))
proof −
have *1*: ⊢ ((($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*)) ∧ *finite*) =
(($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*) ∧ *finite*)
by *auto*
show *?thesis* **by** (*metis 1 int-eq*)
qed

lemma (**in** *Test*) *test5-8-2*:
(*s* ⊨ ▷((($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*)) ∧ *finite*)) =
(*s* ⊨ ▷(($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*) ∧ *finite*))
using *test5-8-1* **unfolding** *Valid-def*
by (*simp add*: *Fstsem*)

lemma (**in** *Test*) *test5-8* :
(*s* ⊨ ▷((($*z*=#*0* ∧ *z gets* $*z*+#*1*) ∧ *fin*($*z*=#*4*)) ∧ *finite*)) =
((*s* ⊨ (($*z* = #*0* ∧ *z gets* $*z* + #*1*) ∧ *fin* ($*z* = #*4*)) ∧ *finite*) ∧ *nlength s* = *0* ∨
0 < *nlength s* ∧
nfinite s ∧
(*s* ⊨ (($*z* = #*0* ∧ *z gets* $*z* + #*1*) ∧ *fin* ($*z* = #*4*)) ∧ *finite*) ∧
(∀ *ia*. *enat ia* < *nlength s* ⟶ (*ntaken ia s* ⊨ ¬ ((($*z* = #*0* ∧ *z gets* $*z* + #*1*) ∧ *fin* ($*z* = #*4*)) ∧
finite))))

using *Fstsem*[*of TEMP* ((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*)) ∧ *finite s*]
by *simp*

lemma (**in** *Test*) *test5-9* :
¬((*s* ⊨ ((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*)) ∧ *finite*) ∧ *nlength s* =*0*)
using *test5-7*
by (*metis* (*mono-tags*, *lifting*) *unl-lift2 zero-neq-numeral*)

lemma (**in** *Test*) *test5-10*:
(*s* ⊨ ((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*)) ∧ *finite*)
⟹
0 < *nlength s* ∧
nfinite s ∧
(*s* ⊨ ((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*) ∧
(∀ *ia*. *enat ia* < *nlength s* ⟶ (*ntaken ia s* ⊨ ¬ (((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*)))

proof −
assume *a0*: *s* ⊨ ((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*)) ∧ *finite*
show *0* < *nlength s* ∧
nfinite s ∧
(*s* ⊨ ((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*) ∧
(∀ *ia*. *enat ia* < *nlength s* ⟶ (*ntaken ia s* ⊨ ¬ (((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*)))
proof −
have *0*: *nfinite s* **using** *a0* **by** (*simp add*: *finite-defs*)
have *1*: *0* < *nlength s* **using** *test5-7 a0 gr-zeroI test5-9* **by** *blast*
have *2*: (∀ *ia*. *enat ia* < *nlength s* ⟶ (*ntaken ia s* ⊨ ¬ (((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*)))
proof
fix *ia*
show *enat ia* < *nlength s* ⟶
(*ntaken ia s* ⊨ ¬ (((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*))
proof −
have *1*: (*ntaken ia s* ⊨ ¬ (((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*)) =
(¬(*ntaken ia s* ⊨ (((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*)))
by *auto*
have *2*: (*ntaken ia s* ⊨ (((\$*z* = #*0* ∧ *z gets* \$*z* + #*1*) ∧ *fin* (\$*z* = #*4*)) ∧ *finite*)) =
(*nlength* (*ntaken ia s*) = *4* ∧ (∀ *i*. *enat i* ≤ *nlength* (*ntaken ia s*) ⟶ *z* (*nnth* (*ntaken ia s*) *i*) = *i*))
using *test5-7*[*of ntaken ia s*] **by** *auto*
have *3*: *enat ia* < *nlength s* ⟶
¬(*nlength* (*ntaken ia s*) = *4* ∧ (∀ *i*. *enat i* ≤ *nlength* (*ntaken ia s*) ⟶ *z* (*nnth* (*ntaken ia s*) *i*) = *i*))
using *a0* **using** *test5-7*[*of ntaken ia s*]
using *not-less-iff-gr-or-eq test5-7* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *blast*
qed
qed

from *0 1 2* **show** *?thesis* **using** *a0* **by** *blast*
qed
qed

lemma (**in** *Test*) *test5-11* :
(*s* ⊨ ▷((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*))) =
(*s* ⊨ ((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*)) ∧ *finite*)
by (*meson test5-10 test5-8 test5-8-0*)

lemma (**in** *Test*) *test5-12* :
⊢ ▷((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*)) = (((\$*z*=#*0* ∧ *z gets* \$*z*+#*1*) ∧ *fin*(\$*z*=#*4*)) ∧ *finite*)
using *test5-11* **by** (*simp add*: *Valid-def*)

end

3.16 Interval Temporal Algebra

We have given an algebraic axiom system for Interval Temporal Logic: Interval Temporal Algebra. The axiom system is a combination of a variant of Kleene algebra and Omega algebra plus axioms for linearity and confluence. Kleene algebra and Omega algebra have been defined by Alasdair Armstrong, Georg Struth and Tjark Weber in [1].

theory *ITA*
imports *Semantics Chopstar Omega*
begin

3.16.1 Definition of Set of intervals and Operations on them

type-synonym *'a iintervals* = *'a nellist set*

definition *lan*:: (*'a*:: *world*) *formula* ⇒ *'a iintervals*
where *lan f* = { σ . (σ ⊨ *f*) }

definition *fusion* :: *'a iintervals* ⇒ *'a iintervals* ⇒ *'a iintervals* (**infixl** · *70*)
where *X*·*Y* = { σ .
(∃ σ*1* σ*2*. σ = *nfuse* σ*1* σ*2* ∧ *nfinite* σ*1* ∧
(σ*1* ∈ *X*) ∧ (σ*2* ∈ *Y*) ∧ (*nlast* σ*1* = *nfirst* σ*2*))
∨ (¬ *nfinite* σ ∧ σ ∈ *X*) }

definition *sempty* :: *'a iintervals* (*SEmpty*)
where
SEmpty ≡ { σ . *nlength* σ = *0* }

definition *smore* :: *'a iintervals* (*SMore*)
where
SMore ≡ − *SEmpty*

definition *sskip* :: *'a iintervals* (*SSkip*)

where
  $SSkip \equiv -(SEmpty \cup (SMore{\cdot}SMore))$

definition $sfalse :: 'a\ iintervals\ (SFalse)$
where
  $SFalse \equiv \{\}$

definition $strue :: 'a\ iintervals\ (STrue)$
where
  $STrue \equiv -\{\}$

definition $sinit :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SInit\ -)\ [85]\ 85)$
where
  $SInit\ X \equiv (X \cap SEmpty){\cdot}STrue$

definition $sinf :: 'a\ iintervals\ (SInf)$
where $SInf \equiv STrue \cdot SFalse$

definition $sfinite ::\ 'a\ iintervals\ (SFinite)$
where $SFinite \equiv -\ SInf$

definition $sfmore :: 'a\ iintervals\ (SFMore)$
where $SFMore \equiv SFinite \cap SMore$

definition $sfin :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SFin\ -)\ [85]\ 85)$
where
  $SFin\ X \equiv SFinite{\cdot}(X \cap SEmpty)$

definition $ssometime :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SSometime\ -)\ [85]\ 85)$
where
  $SSometime\ X \equiv SFinite{\cdot}X$

definition $salways :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SAlways\ -)\ [85]\ 85)$
where
  $SAlways\ X \equiv -(SSometime\ (-X))$

definition $sdi :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SDi\ -)\ [85]\ 85)$
where
  $SDi\ X \equiv X{\cdot}STrue$

definition $sbi :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SBi\ -)\ [85]\ 85)$
where
  $SBi\ X \equiv -(SDi\ (-X))$

definition $sda :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SDa\ -)\ [85]\ 85)$
where
  $SDa\ X \equiv SFinite{\cdot}X{\cdot}STrue$

definition $sba :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SBa\ -)\ [85]\ 85)$
where

$SBa\ X \equiv -(SDa\ (-X))$

definition *snext* :: *'a iintervals* ⇒ *'a iintervals* ((*SNext* -) [*85*] *85*)
where
$SNext\ X \equiv SSkip{\cdot}X$

definition *swnext* :: *'a iintervals* ⇒ *'a iintervals* ((*SWnext* -) [*85*] *85*)
where
$SWnext\ X \equiv (-(SSkip{\cdot}(-X)))$

definition *sprev* :: *'a iintervals* ⇒ *'a iintervals* ((*SPrev* -) [*85*] *85*)
where
$SPrev\ X \equiv X{\cdot}SSkip$

definition *swprev* :: *'a iintervals* ⇒ *'a iintervals* ((*SWprev* -) [*85*] *85*)
where
$SWprev\ X \equiv (-((-X){\cdot}SSkip))$

primrec *spower* :: *'a iintervals* ⇒ *nat* ⇒ *'a iintervals* ((*SPower* - -) [*88*,*88*] *87*)
where
$SPower\ X\ 0 = SEmpty$
| *pwr-Suc*: $SPower\ X\ (Suc\ n) = ((X \cap SFinite){\cdot}(SPower\ X\ n))$

definition *sfpowerstar* :: *'a iintervals* ⇒ *'a iintervals* ((*SFPowerstar* -) [*85*] *85*)
where
$SFPowerstar\ X \equiv (\bigcup\ n.\ SPower\ X\ n)$

definition *spowerstar* :: *'a iintervals* ⇒ *'a iintervals* ((*SPowerstar* -) [*85*] *85*)
where
$SPowerstar\ X \equiv (SFPowerstar\ X){\cdot}(SEmpty \cup (X \cap SInf))$

definition *sstar* :: *'a iintervals* ⇒ *'a iintervals* ((*SStar* -) [*85*] *85*)
where
$SStar\ X \equiv SPowerstar(X \cap SMore)$

3.16.2 Simplification Lemmas

lemma *snot-elim* :
$x \in -X \longleftrightarrow x \notin X$
by *simp*

lemma *sor-elim* :
$x \in (X \cup Y) \longleftrightarrow (x \in X \vee x \in Y)$
by *simp*

lemma *sand-elim* :
$x \in (X \cap Y) \longleftrightarrow (x \in X \wedge x \in Y)$
by *simp*

lemma *sfalse-elim* :

$\sigma \notin SFalse$
by (*simp add*: *sfalse-def*)

lemma *strue-elim* :
$\sigma \in STrue$
by (*simp add*: *strue-def*)

lemma *sempty-elim* :
$\sigma \in SEmpty \longleftrightarrow (\ nlength\ \sigma = 0\ )$
by (*simp add*: *sempty-def*)

lemma *smore-elim* :
$\sigma \in SMore \longleftrightarrow$
$(\ nlength\ \sigma > 0\ )$
by (*simp add*: *sempty-elim smore-def*)

lemma *fusion-iff*:
$\sigma \in X{\cdot}Y \longleftrightarrow$
(
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$(\sigma 1 \in X) \wedge (\sigma 2\ \in Y)\ \wedge (nlast\ \sigma 1 = nfirst\ \sigma 2))$
$\vee\ (\neg\ nfinite\ \sigma \wedge \sigma \in X)\ )$

by (*unfold fusion-def*) *auto*

lemma *fusion-iff-1*:
$\sigma \in X{\cdot}Y \longleftrightarrow$
$(\ (\exists\, n \leq nlength\ \sigma.\ \ (\ (ntaken\ n\ \sigma) \in X\ ) \wedge (\ (ndropn\ n\ \sigma) \in Y)\ )$
$\vee\ (\neg\ nfinite\ \sigma \wedge \sigma \in X)$
)
using *fusion-iff*[*of* σ *X Y*] *nfuse-ntaken-ndropn*[*of - σ*]
by (*simp add*: *chop-nfuse-2*)

lemma *smore-fusion-smore* :
$\sigma \in (SMore{\cdot}SMore) \longleftrightarrow (\ (enat\ 1) < nlength\ \sigma)$
using *fusion-iff-1*[*of* σ *SMore SMore*]
proof (*auto simp add*: *smore-elim*)
show $\bigwedge n\ na.$
$\sigma \in SMore \cdot SMore \Longrightarrow$
$enat\ n \leq nlength\ \sigma \Longrightarrow$
$enat\ n \neq 0 \Longrightarrow$
$nlength\ \sigma - enat\ n \neq 0 \Longrightarrow enat\ na \leq nlength\ \sigma \Longrightarrow enat\ na \neq 0 \Longrightarrow$
$nlength\ \sigma \neq 0 \Longrightarrow nlength\ \sigma - enat\ na \neq 0 \Longrightarrow$
$enat\ (Suc\ 0) < nlength\ \sigma$
by (*metis One-nat-def antisym-conv3 enat-ord-code*(*4*) *idiff-self ileI1 le-less-trans less-le*
not-iless0 one-eSuc one-enat-def)
show $\bigwedge n.\ \sigma \in SMore \cdot SMore \Longrightarrow$
$enat\ n \leq nlength\ \sigma \Longrightarrow$
$enat\ n \neq 0 \Longrightarrow nlength\ \sigma - enat\ n \neq 0 \Longrightarrow$
$\neg\ nfinite\ \sigma \Longrightarrow$

nlength $\sigma \neq 0 \Longrightarrow$
enat (*Suc 0*) $<$ *nlength* σ
by (*metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict zero-enat-def*)
show $\bigwedge n.\ \sigma \in SMore \cdot SMore \Longrightarrow$
$\neg$ *nfinite* $\sigma \Longrightarrow$
enat $n \leq$ *nlength* $\sigma \Longrightarrow$
enat $n \neq 0 \Longrightarrow$
nlength $\sigma \neq 0 \Longrightarrow$
nlength $\sigma -$ *enat* $n \neq 0 \Longrightarrow$
enat (*Suc 0*) $<$ *nlength* σ
by (*metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict zero-enat-def*)
show $\sigma \in SMore \cdot SMore \Longrightarrow \neg$ *nfinite* $\sigma \Longrightarrow$ *nlength* $\sigma \neq 0 \Longrightarrow$ *enat* (*Suc 0*) $<$ *nlength* σ
by (*metis One-nat-def ileI1 linorder-cases nlength-eq-enat-nfiniteD not-less-zero one-eSuc one-enat-def order.not-eq-order-implies-strict*)
show $\sigma \notin SMore \cdot SMore \Longrightarrow$
enat (*Suc 0*) $<$ *nlength* $\sigma \Longrightarrow$
$\forall n.$ *enat* $n \leq$ *nlength* $\sigma \longrightarrow$ *enat* $n = 0 \vee$ *nlength* $\sigma = 0 \vee$ *nlength* $\sigma -$ *enat* $n = 0 \Longrightarrow$
nfinite $\sigma \Longrightarrow$
False
proof $-$
assume *a1*: *enat* (*Suc 0*) $<$ *nlength* σ
assume *nfinite* σ
assume *a2*: $\forall n.$ *enat* $n \leq$ *nlength* $\sigma \longrightarrow$ *enat* $n = 0 \vee$ *nlength* $\sigma = 0 \vee$ *nlength* $\sigma -$ *enat* $n = 0$
have *enat* (*Suc 0*) $= 1$
using *One-nat-def one-enat-def* **by** *presburger*
then have *f3*: *enat 1* $<$ *nlength* σ
using *a1 one-enat-def* **by** *presburger*
have *f4*: *enat 1* $\leq$ *enat 1*
by *blast*
have *enat 1* $\neq \infty$
by *blast*
then show *False*
using *f4 f3 a2*
by (*metis* (*no-types*) *dual-order.strict-iff-order enat-diff-cancel-left gr-implies-not-zero idiff-self one-enat-def zero-neq-one*)
qed
qed

lemma *sskip-elim* :
$\sigma \in SSkip \longleftrightarrow$
(*nlength* $\sigma = 1$)
using *sskip-def smore-fusion-smore*
by (*metis One-nat-def Suc-ile-eq less-numeral-extra*(*4*) *one-enat-def order.not-eq-order-implies-strict sempty-elim smore-def smore-elim snot-elim sor-elim zero-enat-def zero-one-enat-neq*(*1*))

lemma *sinfinite-elim*:
$\sigma \in SInf \longleftrightarrow$
($\neg$ *nfinite* σ)
by (*simp add*: *fusion-iff-1 sfalse-elim sinf-def strue-elim*)

lemma *sfinite-elim*:
$\sigma \in SFinite \longleftrightarrow$
$(nfinite\ \sigma)$
by (*simp add*: *sfinite-def sinfinite-elim*)

lemma *ffusion-iff*:
$\sigma \in (X \cap SFinite)\cdot Y \longleftrightarrow$
(
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$(\sigma 1 \in X) \wedge (\sigma 2\ \in Y)\ \wedge (nlast\ \sigma 1 = nfirst\ \sigma 2))$
)
unfolding *fusion-def*
by (*simp add*: *sfinite-elim*) *blast*

lemma *ffusion-iff-1*:
$\sigma \in (X \cap SFinite)\cdot Y \longleftrightarrow$
$((\ (\exists\, n \leq nlength\ \sigma.\ \ (\ (ntaken\ n\ \sigma) \in X\ ) \wedge (\ (ndropn\ n\ \sigma) \in Y))$
)
)
using *fusion-iff-1*[*of* σ $X \cap SFinite$ Y]
by (*meson nfinite-ntaken sand-elim sfinite-elim*)

lemma *sfmore-elim*:
$\sigma \in SFMore \longleftrightarrow$
$(nfinite\ \sigma \wedge 0 < nlength\ \sigma)$
by (*simp add*: *sfinite-elim sfmore-def smore-elim*)

lemma *spower-elim-zero* :
$\sigma \in SPower\ X\ 0 \longleftrightarrow \sigma \in SEmpty$
by *simp*

lemma *spower-elim-suc* :
$\sigma \in SPower\ X\ (Suc\ n) \longleftrightarrow \sigma \in (X \cap SFinite)\cdot(SPower\ X\ n)$
by *simp*

lemma *spower-elim-suc-1* :
$\sigma \in (X \cap SFinite)\cdot(SPower\ X\ n) \longleftrightarrow$
$(\ (\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$\sigma 1 \in X\ \wedge \sigma 2\ \in (SPower\ X\ n)\ \wedge$
$nlast\ \sigma 1 = nfirst\ \sigma 2\ )$
)
by (*simp add*: *ffusion-iff*)

lemma *ffusionimp*:
assumes $Y0 \subseteq Y1$
shows $\sigma \in (X \cap SFinite)\cdot Y0\ \longrightarrow \sigma \in (X \cap SFinite)\cdot Y1$
using *assms* **unfolding** *ffusion-iff-1* **by** (*auto*)

lemma *spower-finite*:

```
    (SPower X n) ⊆ SFinite
proof (induct n)
case 0
then show ?case
  by (metis nlength-eq-enat-nfiniteD sempty-elim sfinite-elim spower.simps(1) subsetI zero-enat-def)
next
case (Suc n)
then show ?case
  proof −
   have 1: (SPower X (Suc n)) = (X ∩ SFinite)·(SPower X n)
     by simp
   have 2: SPower X n ⊆ SFinite
     using Suc.hyps by blast
   have 3: (X ∩ SFinite)· SFinite ⊆ SFinite
     using ffusion-iff-1[ of - X SFinite]
     by (simp add: sfinite-elim)
        (metis ComplI sfinite-def sinfinite-elim subsetI)
   have 4: (X ∩ SFinite)·(SPower X n) ⊆ (X ∩ SFinite)· SFinite
     using Suc.hyps ffusionimp by blast
   show ?thesis
     using 3 4 by auto
 qed
qed

lemma sfpowerstar-elim:
  σ ∈ SFPowerstar X ⟷ (∃ n. σ ∈ SPower X n)
by (simp add: sfpowerstar-def)

lemma sfpowerstar-elim-1:
  (∃ n. σ ∈ SPower X n) ⟷ ( σ ∈ SPower X 0 ∨ (∃ n. σ ∈ SPower X (Suc n)))
by (metis not0-implies-Suc)

lemma sfpowerstar-suc:
  (∃ n. σ ∈ SPower X (Suc n)) ⟷ (∃ n. σ ∈ (X ∩ SFinite)·(SPower X n))
by simp

lemma sfpowerstar-suc-1:
   (∃ n. σ ∈ (X ∩ SFinite)·(SPower X n)) ⟷ σ ∈ (X ∩ SFinite)·(SFPowerstar X)
unfolding fusion-iff
by (cases σ) (auto simp add: sfpowerstar-elim)

lemma sfpowerstar-eqv-sem:
 σ ∈ SFPowerstar X ⟷ ( σ ∈ SEmpty ∨ σ ∈ (X ∩ SFinite)·(SFPowerstar X))
by (simp add: sfpowerstar-elim sfpowerstar-elim-1 sfpowerstar-suc-1)

lemma sfpowerstar-eqv:
 SFPowerstar X  = SEmpty ∪ (X ∩ SFinite)·(SFPowerstar X)
using sfpowerstar-eqv-sem by blast

lemma sfpowerstar-eqv-1:
```

$(\bigcup\ n.\ SPower\ X\ n) = SEmpty \cup (X \cap SFinite)\cdot(\bigcup\ n.\ SPower\ X\ n)$
using *sfpowerstar-eqv* **by** (*simp add*: *sfpowerstar-def*)

3.16.3 Algebraic Laws

Commutative Additive Monoid

lemma *UnionCommute*:
$(X::'a\ iintervals) \cup Y = Y \cup X$
by (*simp add*: *Un-commute*)

lemma *UnionSFalse*:
$X \cup SFalse = X$
by (*simp add*: *sfalse-def*)

lemma *UnionAssoc*:
$(X::'a\ iintervals) \cup (Y \cup Z) = (X \cup Y) \cup Z$
by (*simp add*: *sup-assoc*)

Boolean algebra

lemma *Huntington*:
$(X::'a\ iintervals) = -(-X \cup -Y) \cup -(-X \cup Y)$
by *auto*

lemma *Morgan*:
$(X::'a\ iintervals) \cap Y = -(-X \cup -Y)$
by *auto*

— identities

lemma *STrueTop*:
$STrue = X \cup -X$
by (*simp add*: *strue-def*)

lemma *SFalseBottom*:
$SFalse = X \cap -X$
by (*simp add*: *sfalse-def*)

multiplicative monoid

lemma *FusionSEmptyLsem*:
$\sigma \in SEmpty \cdot X \longleftrightarrow \sigma \in X$
using *fusion-iff-1*[*of* σ *SEmpty X*]
by (*auto simp add*: *fusion-iff-1 sempty-elim nlength-eq-enat-nfiniteD zero-enat-def*)
(*metis le-numeral-extra*(*3*) *min.bounded-iff min-enat-simps*(*3*) *ndropn-0 zero-enat-def*,
metis le-numeral-extra(*3*) *min.bounded-iff min-enat-simps*(*3*) *ndropn-0 zero-enat-def*)

lemma *FusionSEmptyL* :
$SEmpty \cdot X = X$
using *set-eqI*[*of SEmpty*·*X X*] *FusionSEmptyLsem*

by *auto*

lemma *FusionSEmptyRsem* :
$\sigma \in X \cdot SEmpty \longleftrightarrow \sigma \in X$
using *fusion-iff-1*[*of* σ X *SEmpty*]
by (*auto simp add*: *fusion-iff-1 sempty-elim*)
(*metis is-NNil-ndropn le-numeral-extra*(*3*) *ndropn-0 ndropn-nlength ntaken-all zero-enat-def*,
metis add.right-neutral le-iff-add ndropn-all ndropn-nlength nfinite-nlength-enat nlength-NNil
ntaken-all)

lemma *FusionSEmptyR* :
$X \cdot SEmpty = X$
using *set-eqI*[*of* $X \cdot SEmpty$ X] *FusionSEmptyRsem* **by** *auto*

lemma *FusionAssocA*:
assumes $x \in X \cdot (Y \cdot Z)$
shows $x \in (X \cdot Y) \cdot Z$
proof −
have *1*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge\ \sigma 1 \in X \wedge\ \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2)$
$\vee\ (\neg\ nfinite\ x \wedge x \in X)$
using *assms fusion-iff*[*of* x X $Y \cdot Z$] **by** *auto*
have *2*: $\neg\ nfinite\ x \wedge x \in X \Longrightarrow x \in (X \cdot Y) \cdot Z$
by (*simp add*: *fusion-iff*)
have *3*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge\ \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2) \Longrightarrow$
$x \in (X \cdot Y) \cdot Z$
proof −
assume *a0*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge\ \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2)$
show $x \in (X \cdot Y) \cdot Z$
proof −
obtain $\sigma 1\ \sigma 2$ **where** *5*:
$x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge\ \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2$
using *a0* **by** *auto*
have *6*: $(\exists \sigma 3\ \sigma 4.\ \sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4) \vee$
$(\neg\ nfinite\ \sigma 2 \wedge\ \sigma 2 \in Y)$
using *fusion-iff*[*of* $\sigma 2$ Y Z] *5* **by** *simp*
have *7*: $(\neg\ nfinite\ \sigma 2 \wedge \sigma 2 \in Y) \Longrightarrow x \in (X \cdot Y) \cdot Z$
by (*metis 5 fusion-iff ndropn-nfuse nfinite-ndropn*)
have *8*: $(\exists \sigma 3\ \sigma 4.\ \sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4) \Longrightarrow$
$x \in (X \cdot Y) \cdot Z$
proof −
assume *a1*: $(\exists \sigma 3\ \sigma 4.\ \sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4)$
show $x \in (X \cdot Y) \cdot Z$
proof −
obtain $\sigma 3\ \sigma 4$ **where** *10*:
$\sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge\ \sigma 3 \in Y \wedge\ \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4$
using *a1* **by** *auto*
have *11*: $x = nfuse\ \sigma 1\ (nfuse\ \sigma 3\ \sigma 4)$
using *10 5* **by** *blast*
have *12*: $x = nfuse\ (nfuse\ \sigma 1\ \sigma 3)\ \sigma 4$
by (*simp add*: *10 5 nfirst-nfuse nfuseassoc*)

```
      have 13: (nfuse σ1 σ3) ∈ X·Y
        using fusion-iff[of (nfuse σ1 σ3) X Y]
        using 10 5 nfirst-nfuse by fastforce
      show ?thesis using fusion-iff[of x X·Y Z]
        using 10 12 13 5
        by (metis nfuse-nlength nfinite-nlength-enat nfirst-nfuse nlength-eq-enat-nfiniteD
            plus-enat-simps(1) nlast-nfuse)
      qed
     qed
    show ?thesis
    using 6 7 8 by blast
   qed
  qed
  show ?thesis
  using 1 2 3 by blast
qed

lemma FusionAssocB:
assumes x ∈ (X · Y) · Z
shows   x ∈ X · (Y · Z)
proof −
  have 1: (∃σ1 σ2. x = nfuse σ1 σ2 ∧ nfinite σ1 ∧ σ1 ∈ X · Y ∧ σ2 ∈ Z ∧ nlast σ1 = nfirst σ2) ∨
          (¬ nfinite x ∧ x ∈ X · Y)
    using assms fusion-iff[of x X·Y Z] by auto
  have 2: (¬ nfinite x ∧ x ∈ X · Y) ⟹ x ∈ X · (Y · Z)
    by (metis fusion-iff-1 nfinite-ndropn-b)
  have 3: (∃σ1 σ2. x = nfuse σ1 σ2 ∧ nfinite σ1 ∧ σ1 ∈ X · Y ∧ σ2 ∈ Z ∧ nlast σ1 = nfirst σ2) ⟹
          x ∈ X · (Y · Z)
    proof −
     assume a0: (∃σ1 σ2. x = nfuse σ1 σ2 ∧ nfinite σ1 ∧ σ1 ∈ X · Y ∧ σ2 ∈ Z ∧ nlast σ1 = nfirst σ2)
     show x ∈ X · (Y · Z)
     proof −
      obtain σ1 σ2 where 4:
        x = nfuse σ1 σ2 ∧ nfinite σ1 ∧ σ1 ∈ X · Y ∧ σ2 ∈ Z ∧ nlast σ1 = nfirst σ2
        using a0 by auto
      have 5: ∃σ3 σ4. σ1 = nfuse σ3 σ4 ∧ nfinite σ3 ∧ σ3 ∈ X ∧ σ4 ∈ Y ∧ nlast σ3 = nfirst σ4
        using 4 fusion-iff[of σ1 X Y]
        using nfuse-nappend by fastforce
      obtain σ3 σ4 where 6:
        σ1 = nfuse σ3 σ4 ∧ nfinite σ3 ∧ σ3 ∈ X ∧ σ4 ∈ Y ∧ nlast σ3 = nfirst σ4
        using 5 by auto
      have 7: x = nfuse (nfuse σ3 σ4) σ2
        by (simp add: 4 6)
      have 8: x = nfuse σ3 (nfuse σ4 σ2)
        using 4 6 nfuseassoc by metis
      have 9: (nfuse σ4 σ2) ∈ Y·Z
        using fusion-iff[of (nfuse σ4 σ2) Y Z]
        by (metis 4 6 nfuse-nappend nlast-nfuse)
      have 10: nlast σ3 = nfirst (nfuse σ4 σ2)
        using 4 6 nfirst-nfuse nlast-nfuse by fastforce
```

show *?thesis*
using *fusion-iff*[*of x X Y·Z*]
using *10 6 8 9* **by** *auto*
qed
qed
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *FusionAssoc* :
$X{\cdot}(Y{\cdot}Z) = (X{\cdot}Y){\cdot}Z$
using *set-eqI*[*of* $X{\cdot}(Y{\cdot}Z)$ $(X{\cdot}Y){\cdot}Z$]
FusionAssocA FusionAssocB **by** *blast*

lemma *FusionAssoc1*:
$(X{\cdot}Y){\cdot}Z = X{\cdot}(Y{\cdot}Z)$
by (*simp add*: *FusionAssoc*)

— left and right distributivity

lemma *FusionUnionDistLsem*:
$x{\in}(X \cup Y){\cdot}Z \longleftrightarrow x \in (X{\cdot}Z) \cup (Y{\cdot}Z)$
by (*auto simp add*: *fusion-iff*)

lemma *FusionUnionDistL*:
$(X \cup Y){\cdot}Z = (X{\cdot}Z) \cup (Y{\cdot}Z)$
using *FusionUnionDistLsem*[*of - X Y Z*] **by** *fastforce*

lemma *FusionUnionDistRsem*:
$x{\in}\ X{\cdot}(Y \cup Z) \longleftrightarrow x \in (X{\cdot}Y) \cup (X{\cdot}Z)$
by (*auto simp add*: *fusion-iff*)

lemma *FusionUnionDistR*:
$X{\cdot}(Y \cup Z) = (X{\cdot}Y) \cup (X{\cdot}Z)$
using *FusionUnionDistRsem*[*of - X Y Z*] **by** *fastforce*

— left and right annihilation
lemma *SFalseFusion*:
$SFalse \cdot X = SFalse$
by (*simp add*: *fusion-def sfalse-def*)

lemma *FusionSFalse*:
$X{\cdot}SFalse = (X \cap SInf)$
by (*auto simp add*: *fusion-def sfalse-def sinfinite-elim*)

— idempotency
lemma *UnionIdem*:

$(X::'a\ iintervals) \cup X = X$
by *simp*

Subsumption order

lemma *Subsumption*:
$((X :: 'a\ iintervals) \subseteq Y) = (X \cup Y = Y)$
by *auto*

Helper lemmas

lemma *FusionRuleR*:
assumes $X \subseteq Y$
shows $Z \cdot X \subseteq Z \cdot Y$
using *assms FusionUnionDistR* **by** (*metis Subsumption*)

lemma *FusionRuleL*:
assumes $X \subseteq Y$
shows $X \cdot Z \subseteq Y \cdot Z$
using *assms* **by** (*metis FusionUnionDistL subset-Un-eq*)

lemma *spower-commutes*:
$(X \cap SFinite)\cdot(SPower\ X\ n) = (SPower\ X\ n)\cdot(X \cap SFinite)$
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyL FusionSEmptyR*)
next
case (*Suc n*)
then show *?case* **by** (*simp add*: *FusionAssoc*)
qed

lemma *fusion-inductl*:
assumes $Y \cup X\cdot Z \subseteq Z$
shows $(SPower\ X\ n)\cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyL*)
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: $X \cdot (SPower\ X\ n \cdot Y) \subseteq Z$
using *FusionRuleR Suc.hyps assms* **by** *blast*
have $X \cap SFinite \subseteq X$
by *blast*
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *FusionAssoc FusionRuleL order-trans pwr-Suc*)
qed
qed

lemma *fusion-inductl-finite*:
assumes $Y \cup (X \cap SFinite)\cdot Z \subseteq Z$
shows $(SPower\ X\ n)\cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *FusionSEmptyL*)
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: $SPower\ X\ n \cdot Y \subseteq Z$
using *Suc.hyps assms* **by** *blast*
then have $SPower\ X\ n \cdot Y \cup Z = Z$
by *blast*
then show *?thesis*
using *f1*
by (*metis FusionAssoc1 FusionUnionDistR assms le-supE pwr-Suc*)
qed
qed

lemma *fusion-inductl-fmore*:
assumes $Y \cup (X \cap SFMore)\cdot Z \subseteq Z$
shows $(SPower\ X\ n)\cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *FusionSEmptyL*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: $SPower\ X\ (Suc\ n) \cdot Y = ((X \cap SFinite)\cdot(SPower\ X\ n))\cdot Y$
by *simp*
have *2*: $((X \cap SFinite)\cdot(SPower\ X\ n))\cdot Y = (X \cap SFinite)\cdot((SPower\ X\ n)\cdot Y)$
by (*simp add*: *FusionAssoc*)
have *3*: $(X \cap SFinite)\cdot((SPower\ X\ n)\cdot Y) \subseteq (X \cap SFinite)\cdot Z$
using *FusionRuleR Suc.hyps assms* **by** *blast*
have *31*: $X \cap SFinite = ((X \cap SFMore) \cup ((X \cap SEmpty) \cap SFinite))$
by (*simp add*: *sfmore-def smore-def*) *blast*
have *4*: $(X \cap SFinite)\cdot Z = ((X \cap SFMore)\cdot Z \cup ((X \cap SEmpty) \cap SFinite)\cdot Z)$
by (*simp add*: *31 FusionUnionDistL*)
have *5*: $(X \cap SFMore)\cdot Z \subseteq Z$
using *assms* **by** *auto*
have *6*: $((X \cap SEmpty) \cap SFinite)\cdot Z \subseteq Z$
by (*metis FusionRuleL FusionSEmptyL inf-assoc inf-commute inf-le1*)
show *?thesis*
using *1 2 3 4 5 6* **by** *blast*

qed
qed

lemma *fusion-inductl-inf*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $((SPower\ X\ n) \cdot (X \cap SInf)) \cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case*
by (*metis FusionAssoc FusionRuleL FusionRuleR FusionSEmptyL FusionSFalse le-supE order-trans pwr-0 sfalse-elim subsetI*)
next
case (*Suc n*)
then show *?case*
proof −
have *f2*: $Z = Z \cup (Y \cup X \cdot Z)$
using *assms* **by** *blast*
have $SPower\ X\ n \cdot (X \cap SInf) \cdot Y \subseteq Z$
using *Suc(1) assms* **by** *blast*
then have $Z = Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y$
by *blast*
then have *f3*: $(X \cup X \cap SFinite) \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y) = X \cdot Z$
by *force*
have $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y = X \cap SFinite \cdot (SPower\ X\ n \cdot (X \cap SInf) \cdot Y)$
by (*simp add*: *FusionAssoc*)
then have $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y \cup X \cap SFinite \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y) = X \cap SFinite \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y)$
using *FusionUnionDistR* **by** *blast*
then have $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y \cup X \cdot Z = X \cdot Z$
using *f3 FusionUnionDistL* **by** *blast*
then show *?thesis*
using *f2* **by** *blast*
qed
qed

lemma *fusion-inductl-infmore*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $((SPower\ X\ n) \cdot ((X \cap SMore) \cap SInf)) \cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case*
by (*metis* (*no-types*, *lifting*) *Diff-Compl Int-assoc Un-Diff-Int compl-inf double-compl fusion-inductl-inf inf.absorb-iff2 pwr-0 sfinite-def smore-def spower-finite sup-left-idem*)
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: $\bigwedge n.\ SPower\ X\ n \cdot (X \cap SInf) \cdot Y \subseteq Z$

by (*meson assms fusion-inductl-inf*)
have $X \cap SMore \cap SInf \subseteq X \cap SInf$
by *blast*
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *FusionRuleL FusionRuleR inf.orderE le-inf-iff*)
qed
qed

lemma *fusion-inductl-more*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $(SPower\ (X \cap SMore)\ \ n) \cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyL*)
next
case (*Suc n*)
then show *?case*
by (*metis FusionUnionDistL fusion-inductl sup.boundedE sup.boundedI sup-inf-absorb*)
qed

lemma *fusion-inductr*:
assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot (SPower\ X\ n) \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyR*)
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: $Z \cdot X \subseteq Z$
using *assms* **by** *auto*
have $\forall S.\ Y \cdot (SPower\ X\ n \cdot S) \subseteq Z \cdot S$
using *FusionAssoc FusionRuleL Suc.hyps assms* **by** *blast*
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *FusionRuleR Int-iff order-trans pwr-Suc spower-commutes subsetI*)
qed
qed

lemma *SInfSFinite*:
$X = (X \cap SFinite) \cup (X \cap SInf)$
using *sfinite-def* **by** *auto*

lemma *fusion-inductr-inf*:
assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot ((SPower\ X\ n) \cdot (X \cap SInf)) \subseteq Z$
proof −
have *1*: $Y \cdot (SPower\ X\ n) \subseteq Z$

using *assms fusion-inductr* **by** *blast*
show *?thesis*
proof −
have *f4*: $Y \cdot SPower\ X\ n = Y \cdot SPower\ X\ n \cap Z$
using *1* **by** *blast*
have $Y \cdot SPower\ X\ n \cup Z = Z$
by (*metis 1 subset-Un-eq*)
then have *f5*: $Z \cdot (X \cap SInf) = Z \cdot (X \cap SInf) \cup Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$
using *f4* **by** (*metis* (*no-types*) *FusionUnionDistL sup.commute*)
have $Y \cdot SPower\ X\ n = Y \cdot SPower\ X\ n \cap Z$
using *f4* **by** *auto*
then have $Y \cdot (SPower\ X\ n \cdot (X \cap SInf)) = Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$
by (*simp add*: *FusionAssoc1*)
then show *?thesis*
proof −
have *f1*: $Y \cup Z \cdot X \cup Z = Z$
by (*meson Subsumption assms*)
have *f2*: $Y \cdot SPower\ X\ n \cap Z \cdot X \cup Z \cdot X = Z \cdot X$
by (*metis* (*no-types*) *FusionUnionDistL* ‹$Y \cdot SPower\ X\ n \cup Z = Z$› *f4*)
have $Y \cdot SPower\ X\ n \cap Z \cdot X \cup Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf) = Y \cdot SPower\ X\ n \cap Z \cdot X$
by (*metis* (*no-types*) *FusionUnionDistR sup-inf-absorb*)
then show *?thesis*
using *f2 f1* ‹$Y \cdot (SPower\ X\ n \cdot (X \cap SInf)) = Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$› **by** *auto*
qed
qed
qed

lemma *fusion-inductr-more*:
assumes $Y \cup Z{\cdot}X \subseteq Z$
shows $Y{\cdot}(SPower\ (X \cap SMore)\ \ n) \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyR*)
next
case (*Suc n*)
then show *?case*
by (*metis FusionUnionDistR fusion-inductr le-supE sup.boundedI sup-inf-absorb*)
qed

lemma *FusionUnionSPowersem*:
$\sigma \in Y \cdot (\bigcup n.\ (SPower\ X\ n){\cdot}Z) \longleftrightarrow \sigma \in (\bigcup n.\ Y \cdot ((SPower\ X\ n){\cdot}Z))$
by (*simp add*: *fusion-iff*) *blast*

lemma *FusionUnionSPower*:
$Y \cdot (\bigcup n.\ (SPower\ X\ n){\cdot}Z) = (\bigcup n.\ Y \cdot ((SPower\ X\ n){\cdot}Z))$
using *FusionUnionSPowersem* **by** *blast*

lemma *FusionUnionSPower1*:
$Y \cdot (\bigcup n.\ (SPower\ X\ n)) = (\bigcup n.\ Y \cdot ((SPower\ X\ n)))$

using *FusionUnionSPower*[*of Y X SEmpty*]
by (*metis* (*no-types*, *lifting*) *FusionSEmptyR Sup.SUP-cong*)

lemma *UnionFusionSPowersem*:
$\sigma \in (\bigcup n.\ \ Z{\cdot}(SPower\ X\ n)){\cdot}Y \longleftrightarrow \sigma \in (\bigcup n.\ \ (Z{\cdot}(SPower\ X\ n)){\cdot}Y)$
by (*simp add*: *fusion-iff*) *blast*

lemma *UnionFusionSPower*:
$(\bigcup n.\ \ Z{\cdot}(SPower\ X\ n)){\cdot}Y = (\bigcup n.\ \ (Z{\cdot}(SPower\ X\ n)){\cdot}Y)$
using *UnionFusionSPowersem* **by** *blast*

lemma *UnionFusionSPower1*:
$(\bigcup n.\ \ (SPower\ X\ n)){\cdot}Y = (\bigcup n.\ \ ((SPower\ X\ n)){\cdot}Y)$
using *UnionFusionSPower*[*of SEmpty X Y*]
by (*metis* (*no-types*, *lifting*) *FusionSEmptyL Sup.SUP-cong*)

lemma *spowercommute*:
$(X \cap SFinite){\cdot}(\bigcup n.\ \ (SPower\ X\ n)) = (\bigcup n.\ \ (SPower\ X\ n){\cdot}(X \cap SFinite))$
by (*metis* (*no-types*, *lifting*) *FusionUnionSPower1 Sup.SUP-cong spower-commutes*)

lemma *sstar-contl*:
$Y \cdot (SStar\ X) = (\bigcup n.\ Y \cdot (SPower\ (X \cap SMore)\ n){\cdot}(SEmpty \cup X \cap SMore \cap SInf))$
proof −
have *1*: $Y \cdot (SStar\ X) = Y \cdot ((\bigcup n.\ (\ (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf))$
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: $Y \cdot ((\bigcup n.\ (\ (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)) =$
$(Y \cdot (\bigcup n.\ (SPower\ (X \cap SMore)\ n))){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf)$
using *FusionAssoc* **by** *blast*
have *3*: $(Y \cdot (\bigcup n.\ (SPower\ (X \cap SMore)\ n))) = (\bigcup n.\ Y{\cdot}\ (SPower\ (X \cap SMore)\ n))$
using *FusionUnionSPower1* **by** *blast*
have *4*: $(Y \cdot (\bigcup n.\ (SPower\ (X \cap SMore)\ n))){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf) =$
$(\bigcup n.\ Y{\cdot}\ (SPower\ (X \cap SMore)\ n)){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf)$
by (*simp add*: *3*)
have *5*: $(\bigcup n.\ Y{\cdot}\ (SPower\ (X \cap SMore)\ n)){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf) =$
$(\bigcup n.\ (Y{\cdot}\ (SPower\ (X \cap SMore)\ n)){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf))$
using *UnionFusionSPower*[*of Y X* ∩ *SMore* (*SEmpty* ∪ *X* ∩ *SMore* ∩ *SInf*)]
by *auto*
show *?thesis*
by (*simp add*: *1 2 4 5*)
qed

lemma *sstar-contr*:
$(SStar\ X){\cdot}Y = (\bigcup n.\ ((SPower\ (X \cap SMore)\ n){\cdot}(SEmpty \cup X \cap SMore \cap SInf)){\cdot}Y)$
proof −
have *1*: $(SStar\ X){\cdot}Y = ((\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}(SEmpty \cup X \cap SMore \cap SInf)) \cdot Y$
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: $((\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}(SEmpty \cup X \cap SMore \cap SInf)) \cdot Y =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}((SEmpty \cup X \cap SMore \cap SInf) \cdot Y)$
using *FusionAssoc* **by** *blast*
have *3*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}((SEmpty \cup X \cap SMore \cap SInf) \cdot Y) =$

$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))\cdot((SEmpty \cup X \cap SMore \cap SInf)\cdot\ Y))$
using *UnionFusionSPower1*[*of* $X \cap SMore\ ((SEmpty \cup X \cap SMore \cap SInf)\cdot\ Y)$]
by *auto*
have *4*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))\cdot((SEmpty \cup X \cap SMore \cap SInf)\cdot\ Y)) =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup X \cap SMore \cap SInf))\cdot Y)$
using *FusionAssoc* **by** *blast*
show *?thesis*
by (*simp add*: *1 2 3 4*)
qed

lemma *FPowerstarInductL*:
assumes $Y \cup (X \cap SFinite)\cdot Z \subseteq Z$
shows $(SFPowerstar\ X)\cdot\ Y \subseteq Z$
proof −
have *1*: $(SFPowerstar\ X)\cdot\ Y = (\bigcup n.\ (SPower\ X\ n))\cdot Y$
by (*simp add*: *sfpowerstar-def*)
have *2*: $(\bigcup n.\ (SPower\ X\ n))\cdot Y = (\bigcup n.\ (SPower\ X\ n)\cdot Y)$
using *UnionFusionSPower1* **by** *fastforce*
have *3*: $\bigwedge n.\ (SPower\ X\ n)\cdot Y \subseteq Z$
using *assms fusion-inductl-finite* **by** *blast*
have *4*: $(\bigcup n.\ (SPower\ X\ n)\cdot Y) \subseteq Z$
using *3* **by** *blast*
show *?thesis*
using *1 2 4* **by** *blast*
qed

lemma *FPowerstarInductMoreL*:
assumes $Y \cup ((X \cap SMore) \cap SFinite)\cdot Z \subseteq Z$
shows $(SFPowerstar\ X)\cdot\ Y \subseteq Z$
proof −
have *1*: $(SFPowerstar\ X)\cdot\ Y = (\bigcup n.\ (SPower\ X\ n))\cdot Y$
by (*simp add*: *sfpowerstar-def*)
have *2*: $(\bigcup n.\ (SPower\ X\ n))\cdot Y = (\bigcup n.\ (SPower\ X\ n)\cdot Y)$
using *UnionFusionSPower1* **by** *fastforce*
have *3*: $\bigwedge n.\ (SPower\ X\ n)\cdot Y \subseteq Z$
by (*metis Int-assoc Int-commute assms fusion-inductl-fmore sfmore-def*)
have *4*: $(\bigcup n.\ (SPower\ X\ n)\cdot Y) \subseteq Z$
using *3* **by** *blast*
show *?thesis*
using *1 2 4* **by** *blast*
qed

lemma *PowerstarInductL*:
assumes $Y \cup X\cdot Z \subseteq Z$
shows $(SPowerstar\ X)\cdot Y \subseteq Z$
proof −
have *1*: $(SPowerstar\ X)\cdot\ Y = ((\bigcup n.\ (SPower\ X\ n))\cdot(SEmpty \cup X \cap SInf))\cdot Y$
by (*simp add*: *spowerstar-def sfpowerstar-def*)
have *2*: $((\bigcup n.\ (SPower\ X\ n))\cdot(SEmpty \cup X \cap SInf))\cdot Y =$
$(\bigcup n.\ ((SPower\ X\ n)))\cdot((SEmpty \cup X \cap SInf)\cdot Y)$

using *FusionAssoc1*[*of* $(\bigcup n.\ ((SPower\ X\ n)))\ (SEmpty \cup X \cap SInf)\ Y$]
by *blast*
have *3*: $(\bigcup n.\ ((SPower\ X\ n)))\cdot((SEmpty \cup X \cap SInf)\cdot Y) =$
$(\bigcup n.\ ((SPower\ X\ n))\cdot((SEmpty \cup X \cap SInf)\cdot Y))$
using *UnionFusionSPower1*[*of* $X\ ((SEmpty \cup X \cap SInf)\cdot Y)$]
by *auto*
have *4*: $\bigwedge n.\ (SPower\ X\ n)\cdot Y \subseteq Z$
using *assms fusion-inductl* **by** *blast*
have *5*: $\bigwedge n.\ ((SPower\ X\ n)\cdot(X \cap SInf))\cdot Y \subseteq Z$
using *assms fusion-inductl-inf* **by** *blast*
have *6*: $\bigwedge n.\ (SPower\ X\ n)\cdot Y \cup ((SPower\ X\ n)\cdot(X \cap SInf))\cdot Y =$
$((SPower\ X\ n)\cdot(SEmpty \cup (X \cap SInf)))\cdot Y$
by (*simp add*: *FusionSEmptyR FusionUnionDistL FusionUnionDistR*)
have *7*: $\bigwedge n.\ ((SPower\ X\ n)\cdot(SEmpty \cup (X \cap SInf)))\cdot Y \subseteq Z$
using *4 5 6* **by** *blast*
have *8*: $(\bigcup n.\ ((SPower\ X\ n)\cdot(SEmpty \cup (X \cap SInf)))\cdot Y) \subseteq Z$
using *7* **by** *blast*
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 3 8 FusionAssoc Sup.SUP-cong*)
qed

lemma *SStarInductMoreL*:
assumes $Y \cup (X \cap SMore)\cdot Z \subseteq Z$
shows $(SStar\ X)\cdot Y \subseteq Z$
proof −
have *1*: $(SStar\ X)\cdot\ Y = ((\bigcup n.\ (SPower\ (X \cap SMore)\ n))\cdot(SEmpty \cup (X \cap SMore) \cap SInf))\cdot Y$
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: $((\bigcup n.\ (SPower\ (X \cap SMore)\ \ n))\cdot(SEmpty \cup (X \cap SMore) \cap SInf))\cdot Y =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)))\cdot((SEmpty \cup (X \cap SMore) \cap SInf)\cdot Y)$
using *FusionAssoc1*[*of* $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)))\ (SEmpty \cup (X \cap SMore) \cap SInf)\ Y$]
by *blast*
have *3*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)))\cdot((SEmpty \cup (X \cap SMore) \cap SInf)\cdot Y) =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))\cdot((SEmpty \cup (X \cap SMore) \cap SInf)\cdot Y))$
using *UnionFusionSPower1*[*of* $X \cap SMore\ ((SEmpty \cup (X \cap SMore) \cap SInf)\cdot Y)$]
by *auto*
have *4*: $\bigwedge n.\ (SPower\ (X \cap SMore)\ n)\cdot Y \subseteq Z$
using *assms fusion-inductl* **by** *blast*
have *5*: $\bigwedge n.\ ((SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf))\cdot Y \subseteq Z$
using *assms fusion-inductl-inf* **by** *blast*
have *6*: $\bigwedge n.\ (SPower\ (X \cap SMore)\ n)\cdot Y \cup ((SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf))\cdot Y =$
$((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)))\cdot Y$
by (*simp add*: *FusionSEmptyR FusionUnionDistL FusionUnionDistR*)
have *7*: $\bigwedge n.\ ((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup (\ (X \cap SMore) \cap SInf)))\cdot Y \subseteq Z$
using *4 5 6* **by** *blast*
have *8*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)))\cdot Y) \subseteq Z$
using *7* **by** *blast*
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 3 8 FusionAssoc Sup.SUP-cong*)
qed

lemma *SFPowerstarInductR*:
assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot (SFPowerstar\ X) \subseteq Z$
proof −
have *1*: $Y \cdot (SFPowerstar\ X) = Y \cdot (\bigcup n.\ (SPower\ X\ n))$
by (*simp add*: *sfpowerstar-def*)
have *2*: $Y \cdot (\bigcup n.\ (SPower\ X\ n)) = (\bigcup n.\ Y \cdot (SPower\ X\ n))$
using *FusionUnionSPower1* **by** *blast*
have *3*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have *4*: $(\bigcup n.\ Y \cdot (SPower\ X\ n)) \subseteq Z$
using *3* **by** *blast*
show *?thesis*
by (*simp add*: *1 2 4*)
qed

lemma *SPowerstarInductR*:
assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot (SPowerstar\ X) \subseteq Z$
proof −
have *1*: $Y \cdot (SPowerstar\ X) = Y \cdot ((\bigcup n.\ (SPower\ X\ n)) \cdot\ (SEmpty \cup (X \cap SInf)))$
by (*simp add*: *spowerstar-def sfpowerstar-def*)
have *11*: $Y \cdot ((\bigcup n.\ (SPower\ X\ n)) \cdot\ (SEmpty \cup (X \cap SInf))) =$
$Y \cdot ((\bigcup n.\ (SPower\ X\ n) \cdot\ (SEmpty \cup (X \cap SInf))))$
by (*metis UnionFusionSPower1*)
have *12*: $Y \cdot ((\bigcup n.\ (SPower\ X\ n) \cdot\ (SEmpty \cup (X \cap SInf)))) =$
$((\bigcup n.\ Y \cdot (SPower\ X\ n) \cdot\ (SEmpty \cup (X \cap SInf))))$
using *FusionUnionSPower*[*of Y X* $(SEmpty \cup X \cap SInf)$]
by (*metis* (*no-types*, *lifting*) *FusionAssoc Sup.SUP-cong*)
have *3*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have *31*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \cdot (X \cap SInf) \subseteq Z$
using *FusionAssoc assms fusion-inductr-inf* **by** *blast*
have *32*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf)) \subseteq Z$
by (*simp add*: *3 31 FusionSEmptyR FusionUnionDistR*)
have *4*: $(\bigcup n.\ Y \cdot (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf))) \subseteq Z$
using *32* **by** *blast*
show *?thesis*
by (*simp add*: *1 11 12 4*)
qed

lemma *SStarInductMoreR*:
assumes $Y \cup Z \cdot (X \cap SMore) \subseteq Z$
shows $Y \cdot (SStar\ X) \subseteq Z$
proof −
have *1*: $Y \cdot (SStar\ X) = Y \cdot ((\bigcup n.\ (SPower\ (X \cap SMore)\ n)) \cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf)))$
by (*simp add*: *spowerstar-def sfpowerstar-def sstar-def*)
have *11*: $Y \cdot ((\bigcup n.\ (SPower\ (X \cap SMore)\ n)) \cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf))) =$
$Y \cdot ((\bigcup n.\ (SPower\ (X \cap SMore)\ n) \cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf))))$
by (*metis UnionFusionSPower1*)

have *12*: $Y\cdot((\bigcup n.\ (SPower\ (X \cap SMore)\ n)\cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf)))) =$
$((\bigcup n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf))))$
using *FusionUnionSPower*[*of* Y $(X \cap SMore)$ $(SEmpty \cup ((X \cap SMore) \cap SInf))$]
by (*metis* (*no-types*, *lifting*) *FusionAssoc Sup.SUP-cong*)
have *3*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have *31*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf) \subseteq Z$
using *assms FusionAssoc fusion-inductr-inf* **by** *blast*
have *32*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)) \subseteq Z$
by (*simp add*: *3 31 FusionSEmptyR FusionUnionDistR*)
have *4*: $(\bigcup n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf))) \subseteq Z$
using *32* **by** *blast*
show *?thesis*
by (*simp add*: *1 11 12 4*)
qed

lemma *Powerstarhelp2*:
$(X \cap SInf) = (X \cap SInf)\cdot\ Z$
by (*metis FusionAssoc FusionSFalse SFalseFusion*)

lemma *Powerstarhelp3*:
$((X \cap SInf)\cdot Y \cup (X \cap SFinite)\cdot Y) = X\cdot Y$
by (*metis Diff-Compl FusionUnionDistL Un-Diff-Int sfinite-def*)

lemma *Powerstareqv*:
$(SPowerstar\ X) = SEmpty \cup X\cdot(SPowerstar\ X)$
proof −
have *1*: $(SPowerstar\ X) = (\bigcup n.\ (SPower\ X\ n))\cdot(SEmpty \cup (X \cap SInf))$
by (*simp add*: *sfpowerstar-def spowerstar-def*)
have *2*: $(\bigcup n.\ (SPower\ X\ n))\cdot(SEmpty \cup (X \cap SInf)) =$
$(SEmpty \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n)))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*metis sfpowerstar-eqv-1*)
have *3*: $(SEmpty \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n)))\cdot\ (SEmpty \cup (X \cap SInf)) =$
$SEmpty\cdot(SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*simp add*: *FusionUnionDistL*)
have *4*: $SEmpty\cdot(SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
$=$
$SEmpty \cup (X \cap SInf) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*simp add*: *FusionSEmptyL*)
have *5*: $SEmpty \cup (X \cap SInf) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) =$
$SEmpty \cup (X \cap SInf)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) \cup$
$(X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*metis Powerstarhelp2*)
have *6*: $SEmpty \cup (X \cap SInf)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) \cup$
$(X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) =$
$SEmpty \cup X\ \cdot((\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)))$
by (*metis FusionAssoc Powerstarhelp3 UnionAssoc*)
show *?thesis*
using *1 2 3 4 5 6* **by** *presburger*
qed

lemma *SStareqv*:
SStar X = SEmpty ∪ (X ∩ SMore)·(SStar X)
by (*metis Powerstareqv sstar-def*)

lemma *SSkipSFinite*:
SSkip ∩ SFinite = SSkip
proof −
have *1*: *SSkip ∩ SFinite ⊆ SSkip*
by *simp*
have *2*: *SInf ⊆ SEmpty ∪ (− SEmpty · − SEmpty)*
by (*metis Diff-Compl Diff-subset-conv Powerstarhelp2 Powerstarhelp3 double-compl inf-commute sup-ge1*)

have *3*: *SSkip ⊆ SSkip ∩ SFinite*
using *2* **by** (*simp add*: *sskip-def smore-def sfinite-def*) *blast*
show *?thesis*
using *3* **by** *blast*
qed

lemma *spower-sskip-elim* :
(σ ∈ SPower SSkip n) ⟷
(nlength σ = n)
proof (*induct n arbitrary*: *σ*)
case *0*
then show *?case*
by (*simp add*: *sempty-elim zero-enat-def*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: *(σ ∈ SPower SSkip (Suc n)) ⟷ σ ∈ (SFinite ∩ SSkip)· (SPower SSkip n)*
by (*simp add*: *inf-commute*)
have *2*: *σ ∈ (SFinite ∩ SSkip) ⟷ σ ∈ SSkip*
using *SSkipSFinite* **by** *auto*
have *3*: *σ ∈ (SFinite ∩ SSkip)· (SPower SSkip n) ⟷*
(∃σ1 σ2. σ = nfuse σ1 σ2 ∧ nfinite σ1 ∧ σ1 ∈ SSkip ∧ σ2 ∈ SPower SSkip n ∧ nlast σ1 = nfirst σ2)
using *ffusion-iff*[*of σ SSkip SPower SSkip n*] **by** (*simp add*: *inf-commute*)
have *4*: *(∃σ1 σ2. σ = nfuse σ1 σ2 ∧ nfinite σ1 ∧ σ1 ∈ SSkip ∧ σ2 ∈ SPower SSkip n ∧ nlast σ1 = nfirst σ2) ⟷*
(nlength σ = enat (Suc n))
by (*auto simp add*: *Suc.hyps nfuse-nlength one-enat-def sskip-elim*)
(*metis One-nat-def add.commute eSuc-enat enat.simps*(*3*) *enat-add-sub-same enat-le-plus-same*(*2*)
min.orderE ndropn-nfirst ndropn-nlength nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast
ntaken-nlength one-enat-def plus-1-eSuc(*2*))
show *?thesis*
using *1 3 4* **by** *presburger*
qed
qed

Kleene Algebra

lemma *UnfoldL*:
SEmpty ∪ *X*·(*SStar X*) = (*SStar X*)
proof −
have *1*: (*SStar X*) = *SEmpty* ∪ (*X* ∩ *SMore*)·(*SStar X*)
by (*meson Un-iff set-eqI SStareqv*)
have *2*: (*X* ∩ *SMore*)·(*SStar X*) ⊆ *X*·(*SStar X*)
by (*simp add*: *FusionRuleL*)
have *3*: (*SStar X*) ⊆ *SEmpty* ∪ *X*·(*SStar X*)
using *1 2* **by** *blast*
have *4*: *SEmpty* ⊆ (*SStar X*)
using *1* **by** *auto*
have *5*: *X* ⊆ *SEmpty* ∪ (*X* ∩ *SMore*)
by (*simp add*: *Un-Int-distrib smore-def*)
have *6*: *X*·(*SStar X*) ⊆ (*SStar X*) ∪ (*X* ∩ *SMore*)·(*SStar X*)
using *5* **by** (*metis FusionRuleL FusionUnionDistL FusionSEmptyL*)
have *7*: (*SStar X*) ⊆ *SEmpty* ∪ (*X* ∩ *SMore*)·(*SStar X*)
using *1* **by** *auto*
have *8*: *X*·(*SStar X*) ⊆ *SEmpty* ∪ (*X* ∩ *SMore*)·(*SStar X*)
using *6 7* **by** *blast*
hence *9*: *X*·(*SStar X*) ⊆ (*SStar X*)
using *1* **by** *auto*
have *10*: *SEmpty* ∪ *X*·(*SStar X*) ⊆ (*SStar X*)
using *9 4* **by** *simp*
from *3 10* **show** *?thesis* **by** *auto*
qed

— Left induction

lemma *SStarInductL*:
assumes *Y* ∪ *X*·*Z* ⊆ *Z*
shows (*SStar X*)·*Y* ⊆ *Z*
proof −
have *1*: (*SStar X*)· *Y* = ((⋃ *n*. (*SPower* (*X* ∩ *SMore*) *n*))·(*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*)) ·*Y*
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: ((⋃ *n*. (*SPower* (*X* ∩ *SMore*) *n*))·(*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*)) ·*Y* =
(⋃ *n*. ((*SPower* (*X* ∩ *SMore*) *n*)))·((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*)
using *FusionAssoc1*[*of* (⋃ *n*. ((*SPower* (*X* ∩ *SMore*) *n*))) (*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) *Y*]
by *blast*
have *3*: (⋃ *n*. ((*SPower* (*X* ∩ *SMore*) *n*)))·((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*) =
(⋃ *n*. ((*SPower* (*X* ∩ *SMore*) *n*))·((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*))
using *UnionFusionSPower1*[*of X* ∩ *SMore* ((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*)]
by *auto*
have *31*: *Y* ∪ (*X* ∩ *SMore*)· *Z* ⊆ *Z*
by (*meson FusionRuleL Un-mono assms dual-order.trans inf.cobounded1 subset-refl*)
have *4*: ⋀*n*. (*SPower* (*X* ∩ *SMore*) *n*)·*Y* ⊆ *Z*
using *assms fusion-inductl-more* **by** *blast*
have *5*: ⋀*n*. ((*SPower* (*X* ∩ *SMore*) *n*)·((*X* ∩ *SMore*) ∩ *SInf*))·*Y* ⊆ *Z*
using *31 fusion-inductl-inf* **by** *blast*

have *6*: $\bigwedge n.\ (SPower\ (X \cap SMore)\ n)\cdot Y \cup ((SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf))\cdot Y =$
$((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)))\cdot Y$
by (*simp add*: *FusionSEmptyR FusionUnionDistL FusionUnionDistR*)
have *7*: $\bigwedge n.\ ((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup (\ (X \cap SMore) \cap SInf)))\cdot Y \subseteq Z$
using *4 5 6* **by** *blast*
have *8*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)))\ \cdot Y) \subseteq Z$
using *7* **by** *blast*
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 3 8 FusionAssoc Sup.SUP-cong*)
qed

— Right induction
lemma *SStarInductR*:
assumes $Y \cup Z\cdot X \subseteq Z$
shows $Y\cdot(SStar\ X) \subseteq Z$
proof −
have *1*: $Y\cdot(SStar\ X) = Y\cdot((\bigcup n.\ (SPower\ (X \cap SMore)\ n))\cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf)))$
by (*simp add*: *spowerstar-def sfpowerstar-def sstar-def*)
have *11*: $Y\cdot((\bigcup n.\ (SPower\ (X \cap SMore)\ n))\cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf))) =$
$Y\cdot((\bigcup n.\ (SPower\ (X \cap SMore)\ n)\cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf))))$
by (*metis UnionFusionSPower1*)
have *12*: $Y\cdot((\bigcup n.\ (SPower\ (X \cap SMore)\ n)\cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf)))) =$
$((\bigcup n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot\ (SEmpty \cup ((X \cap SMore) \cap SInf))))$
using *FusionUnionSPower*[*of Y* $(X \cap SMore)$ $(SEmpty \cup ((X \cap SMore) \cap SInf))$]
by (*metis* (*no-types*, *lifting*) *FusionAssoc Sup.SUP-cong*)
have *21*: $Y \cup Z\cdot(X \cap SMore) \subseteq Z$
by (*meson FusionRuleR Un-mono assms dual-order.trans inf.cobounded1 subset-refl*)
have *3*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n) \subseteq Z$
using *21 fusion-inductr* **by** *blast*
have *31*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf) \subseteq Z$
using *21 FusionAssoc fusion-inductr-inf* **by** *blast*
have *32*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)) \subseteq Z$
by (*simp add*: *3 31 FusionSEmptyR FusionUnionDistR*)
have *4*: $(\bigcup n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf))) \subseteq Z$
using *32* **by** *blast*
show *?thesis*
by (*simp add*: *1 11 12 4*)
qed

Omega Algebra

lemma *SFMoresem* [*mono*]:
$x \in (\ (((F \cap SMore) \cap SFinite)\cdot G))\ \longleftrightarrow$
$((\exists \sigma 1\ \sigma 2.\ \ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in F \wedge 0<(nlength\ \sigma 1) \wedge \sigma 2 \in G \wedge nlast\ \sigma 1 = nfirst\ \sigma 2))$

by (*simp add*: *fusion-iff sinfinite-elim smore-elim sfinite-elim*) *blast*

lemma *monoomega*[*mono*]:
mono $(\lambda p\ x.\ \ x \in ((F \cap SMore) \cap SFinite \cdot p))$

by (*auto simp add*: *mono-def sinfinite-elim fusion-iff sfmore-elim*)

coinductive-set *somega* :: *'a iintervals* ⇒ *'a iintervals*
  for *F* **where**
   ((∃ *σ1 σ2*. *x*=*nfuse σ1 σ2* ∧ *nfinite σ1* ∧ *σ1* ∈ *F* ∧ *0*<(*nlength σ1*) ∧
    σ2 ∈ (*somega F*) ∧ *nlast σ1* = *nfirst σ2*))
     ⟹ *x* ∈ (*somega F*)

lemma *SOmegaIntros*:
  (((*F* ∩ *SMore*) ∩ *SFinite*)·(*somega F*)) ⊆ (*somega F*)
using *somega.intros*[*of* - *F*] *SFMoresem*[*of* - *F* (*somega F*)]
by (*meson subsetI*)

lemma *SOmegaCases*:
assumes *x* ∈ *somega F*
      x ∈ ((((*F* ∩ *SMore*) ∩ *SFinite*)·(*somega F*))) ⟹ *P*
 shows *P*
using *assms somega.cases*[*of x F P*] *SFMoresem* **by** *blast*

lemma *SOmegaUnrollsem*:
  x∈ (*somega F*) ⟷ *x*∈ (((*F* ∩ *SMore*) ∩ *SFinite*)·(*somega F*))
proof *auto*
  show *x* ∈ *somega F* ⟹ *x* ∈ (*F* ∩ *SMore*)∩*SFinite* · *somega F*
    using *SOmegaCases* **by** *blast*
  show *x* ∈ (*F* ∩ *SMore*)∩ *SFinite* · *somega F* ⟹ *x* ∈ *somega F*
   by (*metis* (*no-types*, *lifting*) *SOmegaIntros inf-commute* *subset-eq*)
qed

lemma *SOmegaUnroll*:
  (*somega F*) = (((*F* ∩ *SMore*)∩ *SFinite*)·(*somega F*))
using *SOmegaUnrollsem* **by** *blast*

lemma *SOmegaCoinductsem*:
 assumes ⋀ *x*. *x* ∈ *X* ⟹ *x* ∈ ((((*F* ∩ *SMore*)∩*SFinite*)·(*X* ∪ (*somega F*))))
 shows *x* ∈ *X* ⟹ *x* ∈ (*somega F*)
using *assms somega.coinduct*[*of* λ*x*. *x* ∈ *X x F*] *SFMoresem*[*of* - *F* (*X* ∪ *somega F*)] **by** *auto*

lemma *SOmegaCoinduct*:
 assumes *X* ⊆ ((((*F* ∩ *SMore*)∩*SFinite*)·(*X* ∪ (*somega F*))))
 shows *X* ⊆ (*somega F*)
using *assms SOmegaCoinductsem*[*of X F*] **by** *blast*

lemma *SOmegaWeakCoinductsem*:
 assumes ⋀ *x*. *x* ∈ *X* ⟹ *x* ∈ ((((*F* ∩ *SMore*)∩*SFinite*)·*X*))
 shows *x* ∈ *X* ⟹ *x* ∈ (*somega F*)
using *assms somega.coinduct*[*of* λ*x*. *x* ∈ *X x F*]
by (*metis SFMoresem*)

lemma *SOmegaWeakCoinduct*:
 assumes *X* ⊆ ((((*F* ∩ *SMore*)∩*SFinite*)·*X*))

shows $X \subseteq (somega\ F)$
using *assms SOmegaWeakCoinductsem*[*of X F*] **by** *blast*

ITL specific Laws

lemma *PwrFusionInterLsem*:
$x \in ((((SPower\ SSkip\ n) \cap X)\cdot V) \cap (((SPower\ SSkip\ n) \cap Y)\cdot W)) \longleftrightarrow$
$x \in (((SPower\ SSkip\ n) \cap X \cap Y)\cdot(V \cap W))$
by (*auto simp add*: *spower-sskip-elim fusion-iff-1 min-absorb1 nlength-eq-enat-nfiniteD*)

lemma *PwrFusionInterL*:
$((((SPower\ SSkip\ n) \cap X)\cdot V) \cap (((SPower\ SSkip\ n) \cap Y)\cdot W)) =$
$(((SPower\ SSkip\ n) \cap X \cap Y)\cdot(V \cap W))$
using *PwrFusionInterLsem* **by** *blast*

lemma *PwrFusionInterRsem* :
$x \in ((V\cdot((SPower\ SSkip\ n) \cap X)) \cap ((W\cdot((SPower\ SSkip\ n) \cap Y)))) \longleftrightarrow$
$x \in ((V \cap W)\cdot((SPower\ SSkip\ n) \cap X \cap Y))$
by (*simp add*: *spower-sskip-elim fusion-iff-1*)
(*rule*,
metis enat.simps(*3*) *enat-add2-eq enat-add-sub-same le-iff-add ndropn-nlength nfinite-ndropn*
nlength-eq-enat-nfiniteD the-enat.simps,
metis enat.simps(*3*) *enat-add-sub-same le-iff-add ndropn-nlength nfinite-ndropn-b*
nlength-eq-enat-nfiniteD)

lemma *PwrFusionInterR* :
$((V\cdot((SPower\ SSkip\ n) \cap X)) \cap ((W\cdot((SPower\ SSkip\ n) \cap Y)))) =$
$((V \cap W)\cdot((SPower\ SSkip\ n) \cap X \cap Y))$
using *PwrFusionInterRsem* **by** *blast*

lemma *SSkipFusionImpSMore*:
$SSkip\cdot STrue \subseteq SMore$
using *subsetI*[*of SSkip·STrue SMore*]
by (*auto simp add*: *fusion-iff-1 sskip-elim smore-elim strue-elim*)

lemma *SMoreImpSSkipFusion*:
$SMore \subseteq SSkip\cdot STrue$
using *subsetI*[*of SMore SSkip·STrue*]
by (*auto simp add*: *fusion-iff-1 sskip-elim smore-elim strue-elim*)
(*metis i0-less ileI1 one-eSuc one-enat-def*,
metis i0-less ileI1 one-eSuc one-enat-def)

lemma *SSkipSMore*:
$SSkip \cap SMore = SSkip$
by (*simp add*: *inf.absorb1 smore-def sskip-def*)

lemma *SPowerSkip*:
$(\bigcup n.\ (SPower\ SSkip\ n)) = SFinite$
proof −
have *1*: $(\bigcup n.\ (SPower\ SSkip\ n)) \subseteq SFinite$

by (*simp add*: *SUP-least spower-finite*)
have *2*: $\bigwedge x.\ \ x \in SFinite \Longrightarrow x \in (\bigcup n.\ (SPower\ SSkip\ n))$
by (*auto simp add*: *sfinite-elim spower-sskip-elim nfinite-nlength-enat*)
have *3*: $SFinite \subseteq (\bigcup n.\ (SPower\ SSkip\ n))$
using *2* **by** *blast*
from *1 3* **show** *?thesis* **by** *blast*
qed

lemma *SStarSkip*:
$(SStar\ SSkip) = SFinite$
by (*simp add*: *sstar-def SSkipSMore spowerstar-def sfpowerstar-def SPowerSkip*)
(*metis Compl-disjoint2 FusionSEmptyR SSkipSFinite UnionSFalse inf.assoc inf-bot-right sfalse-def sfinite-def*)

3.16.4 Derived Laws

Helper Lemmas

lemma *B01*:
assumes $(X::\ 'a\ iintervals) \subseteq Y$
shows $-Y \subseteq -X$
using *assms* **by** *auto*

lemma *B04*:
$((X::\ 'a\ iintervals) = Y\ ) \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$
by *auto*

lemma *B09*:
assumes $-X \cup Y = STrue$
shows $(X::\ 'a\ iintervals) \subseteq Y$
using *assms* **using** *strue-def* **by** *auto*

lemma *B20*:
$(X::\ 'a\ iintervals) \subseteq Y \cup Z \longleftrightarrow X \cap -Y \subseteq Z$
by *auto*

lemma *B28*:
$((X::\ 'a\ iintervals) \cap Y) \cup (X \cap Z) = X \cap (Y \cup Z)$
by *auto*

lemma *CH01*:
$STrue{\cdot}STrue = STrue$
by (*metis FusionSEmptyR FusionUnionDistR Int-commute STrueTop inf-sup-absorb*)

lemma *CH07*:
$(((SSkip \cap X){\cdot}V) \cap ((SSkip \cap Y){\cdot}W)) = ((SSkip \cap X \cap Y){\cdot}(V \cap W))$
using *PwrFusionInterL*[*of 1 X V Y W*]
by (*simp add*: *FusionSEmptyR SSkipSFinite*)

lemma *CH08*:
$((V{\cdot}(SSkip \cap X)) \cap ((W{\cdot}(SSkip \cap Y)))) = ((V \cap W){\cdot}(SSkip \cap X \cap Y))$

using *PwrFusionInterR*[*of V 1 X W Y*]
by (*simp add*: *FusionSEmptyR SSkipSFinite*)

lemma *CH09*:
$(((X \cap SEmpty)\cdot V) \cap ((Y \cap SEmpty)\cdot W)) = (((X \cap Y) \cap SEmpty)\cdot(V \cap W))$
using *PwrFusionInterL*[*of 0 X V Y W*]
by (*metis* (*no-types*, *lifting*) *inf-assoc inf-commute pwr-0*)

lemma *CH10*:
$((V\cdot(X \cap SEmpty)) \cap ((W\cdot(Y \cap SEmpty)))) = ((V \cap W)\cdot((X \cap Y) \cap SEmpty))$
using *PwrFusionInterR*[*of V 0 X W Y*]
by (*metis* (*no-types*, *lifting*) *inf-assoc inf-commute pwr-0*)

lemma *ST13*:
$((X \cap SEmpty)\cdot Z) \cap ((Y \cap SEmpty)\cdot Z) = ((X \cap Y) \cap SEmpty)\cdot Z$
by (*simp add*: *CH09*)

lemma *ST15*:
$(SStar\ (X \cap SEmpty)) = SEmpty$
using *SStareqv*[*of* $(X \cap SEmpty)$]
by (*metis Compl-disjoint2 Powerstarhelp2 SFalseBottom UnionSFalse inf-assoc inf-bot-right sfalse-def smore-def*)

lemma *ST21*:
$((-X) \cap SEmpty) \cup (X \cap SEmpty) = SEmpty$
by *blast*

lemma *ST24*:
$(SInit\ X) \cap (SInit\ Y) = (SInit\ (X \cap Y))$
by (*simp add*: *ST13 sinit-def*)

lemma *ST25*:
$(SInit\ STrue) = STrue$
by (*simp add*: *sinit-def strue-def FusionSEmptyL*)

lemma *ST26*:
$(SInit\ (-X)) \cup (SInit\ X) = STrue$
by (*metis Compl-disjoint2 ST21 ST25 Un-Int-distrib compl-bot-eq FusionUnionDistL sinit-def strue-def sup-bot.right-neutral sup-top-right*)

lemma *ST28*:
$(SDi\ (SInit\ X)) = (SInit\ X)$
by (*metis compl-bot-eq FusionAssoc FusionUnionDistR FusionSEmptyR sdi-def sinit-def strue-def sup-top-right UnionCommute*)

lemma *ST33*:
$(STrue \cap SEmpty)\cdot SEmpty = SEmpty$
by (*simp add*: *strue-def FusionSEmptyL*)

lemma *ST36*:

$(SInit\ (-X)) \subseteq -(SInit\ X)$
unfolding *sinit-def*
by (*metis SFalseBottom SFalseFusion ST24 disjoint-eq-subset-Compl inf.commute inf-compl-bot-left2 sinit-def*)

lemma *ST37*:
$-(SInit\ X) \subseteq (SInit\ (-X))$
using *B09 ST26* **by** *auto*

lemma *ST38*:
$-(SInit\ X) = (SInit\ (-X))$
using *ST37 ST36* **by** *auto*

lemma *ST47*:
$X \cup Y{\cdot}X = (SEmpty \cup Y){\cdot}X$
by (*simp add*: *FusionUnionDistL FusionSEmptyL*)

lemma *SStar01*:
assumes $X{\cdot}(SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$
shows $(SStar\ X) \subseteq (SStar\ Y)$
using *assms*
by (*metis Un-commute FusionSEmptyR SStarInductL*)

lemma *SStar03*:
$(SStar\ X){\cdot}(SStar\ X) \subseteq (SStar\ X)$
by (*metis SStarInductL Un-absorb UnfoldL sup.absorb-iff1 sup.right-idem*)

lemma *SStar05*:
assumes $((SStar\ X){\cdot}(SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$
shows $(SStar\ (SStar\ X)) \subseteq (SStar\ X)$
using *assms*
by (*simp add*: *SStar01*)

lemma *SStar12*:
$(SEmpty \cup (X{\cdot}(SStar\ X))) \subseteq (SStar\ X)$
using *UnfoldL* **by** *blast*

lemma *SStar06*:
$((SStar\ X){\cdot}(SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$
using *SStar03 SStar12* **by** *force*

lemma *SStar07*:
$(SStar\ X) \subseteq (SStar\ (SStar\ X))$
proof −
have $SStar\ X \cdot SEmpty \cup SStar\ X \cdot (SStar\ X \cdot SStar\ SStar\ X) = SStar\ X \cdot SStar\ SStar\ X$
by (*metis* (*full-types*) *FusionUnionDistR UnfoldL*)
then have $SStar\ X \cdot SEmpty \cup SStar\ X \cdot (SStar\ X \cdot SStar\ SStar\ X) \subseteq SStar\ SStar\ X$
using *UnfoldL* **by** *auto*
then show *?thesis*
by (*simp add*: *FusionSEmptyR*)

qed

lemma *SStar08*:
$(SStar\ X) = (SStar\ (SStar\ X))$
by (*meson B04 SStar05 SStar06 SStar07*)

lemma *SStar15*:
$SEmpty \subseteq (SStar\ SSkip)$
using *UnfoldL* **by** *fastforce*

lemma *SStar16*:
$SSkip \subseteq (SStar\ SSkip)$
using *SSkipSFinite SStarSkip* **by** *blast*

lemma *SStar22*:
$(SEmpty \cap X)\cdot(SStar\ (SEmpty \cap X)) = (SEmpty \cap X)$
by (*metis ST15 FusionSEmptyR inf-commute*)

lemma *SStar23*:
$(SStar\ (SEmpty \cap X)) = SEmpty$
using *SStar22 UnfoldL* **by** *auto*

lemma *SStar25*:
$(SStar\ STrue) = STrue$
by (*metis FusionRuleR FusionSEmptyR Un-absorb2 UnfoldL UnionCommute compl-bot-eq strue-def sup.right-idem sup-ge2 sup-top-right*)

lemma *SStar28*:
$(SStar\ X)\cdot X \subseteq X\cdot(SStar\ X)$
by (*metis B04 FusionUnionDistR FusionSEmptyR UnfoldL SStarInductL*)

lemma *SStar29*:
$X\cdot(SStar\ X) \subseteq (SStar\ X)\cdot X$
by (*metis B04 SStar28 SStarInductR UnfoldL FusionRuleL ST47 sup.mono*)

lemma *SStar17*:
$(SStar\ SSkip)\cdot SSkip \subseteq SSkip\cdot(SStar\ SSkip)$
by (*simp add*: *SStar28*)

lemma *SStar18*:
$SSkip\cdot(SStar\ SSkip) \subseteq (SStar\ SSkip)\cdot SSkip$
by (*simp add*: *SStar29*)

lemma *SStar19*:
$(SStar\ SSkip)\cdot SSkip = SSkip\cdot(SStar\ SSkip)$
using *SStar17 SStar18* **by** *auto*

lemma *SStar30*:
$X\cdot(SStar\ X) = (SStar\ X)\cdot X$
using *SStar28 SStar29* **by** *auto*

lemma *SStar34*:
assumes $SEmpty \cup (X \cup Y)\cdot((SStar\ X)\cdot(SStar\ (Y\cdot(SStar\ X)))\ ) \subseteq (SStar\ X)\cdot(SStar\ (Y\cdot(SStar\ X)))$
shows $(SStar\ (X \cup Y)) \subseteq (SStar\ X)\cdot(SStar\ (Y\cdot(SStar\ X)))$
by (*metis assms FusionSEmptyR SStarInductL*)

lemma *SStar35*:
$SEmpty \cup (X \cup Y)\cdot((SStar\ X)\cdot(SStar\ (Y\cdot(SStar\ X)))\ ) \subseteq (SStar\ X)\cdot(SStar\ (Y\cdot(SStar\ X)))$
by (*simp add*: *FusionAssoc FusionUnionDistL ST47 UnfoldL UnionAssoc UnionCommute*)

lemma *SStar36*:
$(SStar\ (X \cup Y)) \subseteq (SStar\ X)\cdot(SStar\ (Y\cdot(SStar\ X)))$
using *SStar34 SStar35* **by** *blast*

lemma *SStar46*:
$(SStar\ X)\cdot(SStar\ (Y\cdot(SStar\ X))) \subseteq (SStar\ (X \cup Y))$
proof −
have $(SEmpty \cup SStar\ (X \cup Y)\ \cdot\ Y)\ \cdot\ SStar\ X \subseteq SStar\ (X \cup Y)$
by (*metis* (*no-types*) *FusionUnionDistR SStar12 SStar30 SStarInductR sup.bounded-iff*)
then show *?thesis* **by** (*simp add*: *SStarInductR ST47 FusionAssoc*)
qed

lemma *SStar47*:
$(SStar\ Z) = (SStar\ (Z \cap SMore))$
proof −
have *1*: $(SStar\ Z) = (SStar\ ((SEmpty \cap Z) \cup (SMore \cap Z)))$
by (*metis Int-Un-distrib2 compl-bot-eq inf-top.left-neutral smore-def strue-def STrueTop*)
have *2*: $(SStar\ ((SEmpty \cap Z) \cup (SMore \cap Z))) =$
$(SStar\ (SEmpty \cap Z))\cdot(SStar\ ((SMore \cap Z)\cdot(SStar\ (SEmpty \cap Z))))$
by (*simp add*: *SStar36 SStar46 subset-antisym*)
have *3*: $(SStar\ (SEmpty \cap Z))\cdot(SStar\ ((SMore \cap Z)\cdot(SStar\ (SEmpty \cap Z)))) =$
$(SStar\ (Z \cap SMore))$
by (*simp add*: *FusionSEmptyL FusionSEmptyR SStar23 inf-commute*)
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *SStar48*:
$(SStar\ SMore) = STrue$
by (*metis Compl-Un Compl-disjoint2 SStar25 SStar47 ST21 ST33 FusionSEmptyR*
inf.right-idem smore-def strue-def)

lemma *SStar50*:
assumes $SSkip\cdot((-X) \cup ((SStar\ SSkip)\cdot(X \cap (SSkip\cdot(-X))))) \cup (-X)$
$\subseteq (-X) \cup (SStar\ SSkip)\cdot(X \cap (SSkip\cdot(-X)))$
shows $((SStar\ SSkip)\cdot(-X)) \subseteq ((-X) \cup ((SStar\ SSkip)\cdot(X \cap (SSkip\cdot(-X)))))$
using *SStarInductL assms* **by** *blast*

lemma *SStar51*:
$SSkip\cdot((-X) \cup ((SStar\ SSkip)\cdot(X \cap (SSkip\cdot(-X))))) \cup (-X)$
$\subseteq (-X) \cup (SStar\ SSkip)\cdot(X \cap (SSkip\cdot(-X)))$

using *FusionUnionDistR*[*of SSkip*] *UnfoldL*[*of SSkip*]
proof −
have *f1*: $-\ X \cup (SEmpty \cup SSkip \cdot SStar\ SSkip) \cdot (X \cap (SSkip \cdot -\ X)) \subseteq$
$-\ X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -\ X))$
by (*simp add*: ‹*SEmpty* ∪ *SSkip* · *SStar SSkip* = *SStar SSkip*›)
have *f2*: $SSkip \cdot -\ X \subseteq -\ X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -\ X))$
by (*metis B20 Morgan ST47 UnionIdem* ‹*SEmpty* ∪ *SSkip* · *SStar SSkip* = *SStar SSkip*›
inf-commute inf-idem sup.cobounded1)
have $SSkip \cdot (SStar\ SSkip \cdot (X \cap (SSkip \cdot -\ X))) \subseteq -\ X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -\ X))$
using *f1* **by** (*metis* (*no-types*) *FusionAssoc ST47 Un-subset-iff*)
then show *?thesis*
using *f2* **by** (*simp add*: ‹$\bigwedge Z\ Y.\ SSkip \cdot (Y \cup Z) = SSkip \cdot Y \cup SSkip \cdot Z$›)
qed

lemma *SStar52*:
$(SStar\ X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$
by (*metis B04 SStar47 UnfoldL*)

lemma *SStar53*:
$SEmpty \cup (X \cap SMore) \cdot (SStar\ X) \subseteq (SStar\ X)$
by (*metis SStar12 SStar47*)

lemma *BD45*:
$(SBi\ ((-X) \cup X1)) \cap (X \cdot Y) \subseteq X1 \cdot Y$
proof −
have *1*: $(SBi\ ((-X) \cup X1)) = -(-((-X) \cup X1) \cdot (Y \cup (-Y)))$
by (*metis sbi-def sdi-def STrueTop*)
have *2*: $-(-((-X) \cup X1) \cdot (Y \cup (-Y))) \cap (X \cdot Y) \subseteq$
$-(-((-X) \cup X1) \cdot (Y)) \cap (X \cdot Y)$
using *FusionUnionDistR* **by** *fastforce*
have *3*: $-(-((-X) \cup X1) \cdot (Y)) \cap (X \cdot Y) \subseteq\ (((-X) \cup X1) \cap X) \cdot Y$
by (*metis* (*no-types*, *opaque-lifting*) *B20 FusionRuleL FusionUnionDistL Morgan UnionCommute*
double-compl order-refl)
have *4*: $(((-X) \cup X1) \cap X) \cdot Y \subseteq X1 \cdot Y$
by (*metis B20 double-compl FusionRuleL inf.right-idem inf-le1*)
from *1 2 3 4* **show** *?thesis* **by** *blast*
qed

lemma *BD46*:
$(SAlways\ ((-Y) \cup\ Y1)) \cap (X1 \cdot Y) \subseteq (X1 \cdot Y1)$
proof −
have *1*: $(SAlways\ ((-Y) \cup\ Y1)) = -((SFinite \cap (X1 \cup -X1)) \cdot\ (-((-Y) \cup\ Y1\ )\ )\ )$
by (*simp add*: *salways-def ssometime-def*)
have *2*: $-((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup\ Y1\ )\ )\ ) =$
$-((\ (SFinite \cap X1) \cup (SFinite \cap -X1)) \cdot (-((-Y) \cup\ Y1\ )\ )\ )$
by (*simp add*: *B28*)
have *3*: $-((\ (SFinite \cap X1) \cup (SFinite \cap -X1)) \cdot (-((-Y) \cup\ Y1\ )\ )\ ) =$
$-(\ (SFinite \cap X1) \cdot (-((-Y) \cup\ Y1\ )) \cup (SFinite \cap -X1) \cdot (-((-Y) \cup\ Y1\ ))\ )$
by (*simp add*: *FusionUnionDistL*)
have *4*: $-(\ (SFinite \cap X1) \cdot (-((-Y) \cup\ Y1\ )) \cup (SFinite \cap -X1) \cdot (-((-Y) \cup\ Y1\ ))\ ) =$

$-((SFinite \cap X1)\cdot(-((-Y) \cup Y1\ ))) \cap -((SFinite \cap -X1)\cdot(-((-Y) \cup Y1\ )))$
by *blast*
have *5*: $(X1\cdot Y) = ((SFinite \cap X1)\cdot Y) \cup (SInf \cap X1\ )$
by (*metis Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute*)
have *6*: $-((SFinite \cap X1)\cdot(-((-Y) \cup Y1\ ))) =$
$-((SFinite \cap X1)\cdot(Y \cap -Y1\ ))$
by *simp*
have *7*: $-((SFinite \cap X1)\cdot(-((-Y) \cup Y1\ ))\ ) \cap ((SFinite \cap X1)\cdot Y) \subseteq$
$(SFinite \cap X1)\cdot(\ ((-Y) \cup Y1) \cap Y)$
by (*metis* (*no-types*) *B20 FusionRuleR FusionUnionDistR double-compl inf-sup-aci*(*1*) *order-refl*)
have *8*: $(SFinite \cap X1)\cdot(\ ((-Y) \cup Y1) \cap Y) \subseteq (SFinite \cap X1)\cdot Y1$
by (*metis B20 FusionRuleR double-compl inf.cobounded1 inf.right-idem*)
have *9*: $(SFinite \cap X1)\cdot Y1 \subseteq X1 \cdot Y1$
by (*simp add*: *FusionRuleL*)
have *10*: $-((SFinite \cap X1)\cdot(-((-Y) \cup Y1\ ))\ ) \cap (SInf \cap X1) \subseteq ((SFinite \cap X1)\cdot Y1) \cup (SInf \cap X1)$
by *blast*
have *11*: $((SFinite \cap X1)\cdot Y1) \cup (SInf \cap X1) \subseteq X1 \cdot Y1$
by (*metis B04 Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute*)
have *12*: $-((SFinite \cap (X1 \cup -X1))\cdot\ (-((-Y) \cup Y1\ )\ )\ ) \cap (X1\cdot Y) =$
$(-((SFinite \cap (X1 \cup -X1))\cdot\ (-((-Y) \cup Y1\ )\ )\ ) \cap ((SFinite \cap X1)\cdot Y)) \cup$
$(-((SFinite \cap (X1 \cup -X1))\cdot\ (-((-Y) \cup Y1\ )\ )\ ) \cap (SInf \cap X1))$
by (*simp add*: *5 B28*)
have *13*: $(-((SFinite \cap (X1 \cup -X1))\cdot\ (-((-Y) \cup Y1\ )\ )\ ) \cap ((SFinite \cap X1)\cdot Y)) \subseteq X1 \cdot Y1$
using *7 8 9 2 3* **by** *blast*
have *14*: $(-((SFinite \cap (X1 \cup -X1))\cdot\ (-((-Y) \cup Y1\ )\ )\ ) \cap (SInf \cap X1)) \subseteq\ X1 \cdot Y1$
using *10 11* **by** *blast*
have *15*: $(-((SFinite \cap (X1 \cup -X1))\cdot\ (-((-Y) \cup Y1\ )\ )\ ) \cap ((SFinite \cap X1)\cdot Y)) \cup$
$(-((SFinite \cap (X1 \cup -X1))\cdot\ (-((-Y) \cup Y1\ )\ )\ ) \cap (SInf \cap X1)) \subseteq X1 \cdot Y1$
using *13 14* **by** *blast*
show *?thesis*
using *1 12 15* **by** *blast*
qed

ITL Axioms derived

lemma *SBoxGen*:
assumes $X{=}STrue$
shows $(SAlways\ X) = STrue$
using *assms*
by (*metis Compl-disjoint2 FusionSFalse double-compl salways-def sfalse-def sfinite-def ssometime-def strue-def*)

lemma *SBiGen*:
assumes $X{=}STrue$
shows $(SBi\ X) = STrue$
using *assms*
by (*metis double-compl sbi-def sdi-def sfalse-def SFalseFusion strue-def*)

lemma *SMP*:
assumes $X \subseteq Y$

assumes $X{=}STrue$
shows $Y{=}STrue$
using *assms* **using** *strue-def* **by** *blast*

lemma *SChopAssoc*:
$X{\cdot}(Y{\cdot}Z) = (X{\cdot}Y){\cdot}Z$
by (*simp add*: *FusionAssoc*)

lemma *SOrChopImp*:
$(X \cup Y){\cdot}Z \subseteq (X{\cdot}Z) \cup (Y{\cdot}Z)$
by (*simp add*: *FusionUnionDistL*)

lemma *SChopOrImp*:
$X{\cdot}(Y \cup Z) \subseteq (X{\cdot}Y) \cup (X{\cdot}Z)$
by (*simp add*: *FusionUnionDistR*)

lemma *SEmptyChop*:
$SEmpty{\cdot}X = X$
by (*simp add*: *FusionSEmptyL*)

lemma *SChopEmpty*:
$X{\cdot}SEmpty = X$
by (*simp add*: *FusionSEmptyR*)

lemma *SStateImpBi*:
$(SInit\ X) \subseteq (SBi\ (SInit\ X))$
by (*simp add*: *ST28 ST38 sbi-def*)

lemma *SNextImpNotNextNot*:
$(SNext\ X) \subseteq -(SNext\ (-X))$
proof −
have *1*: $((SNext\ X) \subseteq -(SNext\ (-X))) = (((SNext\ X) \cap (SNext\ (-X))) \subseteq SFalse)$
by (*simp add*: *disjoint-eq-subset-Compl sfalse-def*)
have *2*: $((SNext\ X) \cap (SNext\ (-X))) = SSkip{\cdot}(X \cap (-X))$
by (*metis CH07 SStar16 inf.orderE snext-def*)
have *3*: $(SSkip){\cdot}(X \cap (-X)) = SSkip{\cdot}SFalse$
by (*simp add*: *sfalse-def*)
have *4*: $SSkip{\cdot}SFalse = SFalse$
using *FusionSFalse SStar16 SStarSkip sfalse-def sfinite-def* **by** *fastforce*
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *SBiBoxChopImpChop*:
$(SBi\ ((-X) \cup X1)) \cap (SAlways\ ((-Y) \cup Y1)) \cap (X{\cdot}Y) \subseteq (X1{\cdot}Y1)$
using *BD45 BD46* **by** *blast*

lemma *SBoxInduct*:
$(SAlways\ (-X \cup (SWnext\ X))) \cap X \subseteq (SAlways\ X)$
proof −
have *1*: $((SAlways\ (-X \cup (SWnext\ X))) \cap X \subseteq (SAlways\ X)) =$

$((SSometime\ (-X)) \subseteq ((-X) \cup (SSometime\ (X \cap (SNext\ (-X)))\ )))$
by (*simp add*: *salways-def snext-def swnext-def*)
blast
have *2*: $((SSometime\ (-X)) \subseteq ((-X) \cup (SSometime\ (X \cap (SNext\ (-X)))\ ))) =$
$(\ ((SStar\ SSkip)\cdot(-X)) \subseteq ((-X) \cup ((SStar\ SSkip)\cdot(X \cap (SSkip\cdot(-X))))))\ \ )$
by (*simp add*: *SStarSkip snext-def ssometime-def*)
have *3*: $(\ ((SStar\ SSkip)\cdot(-X)) \subseteq ((-X) \cup ((SStar\ SSkip)\cdot(X \cap (SSkip\cdot(-X))))))\ \ )$
using *SStar51 SStar50* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *SChopstarEqv*:
$(SStar\ X) = SEmpty \cup (X \cap SMore)\cdot(SStar\ X)$
using *SStar52 SStar53* **by** *blast*

3.16.5 Extra Laws

Boolean Laws

lemma *B02*:
assumes $-Y \subseteq -X$
shows $(X{::}\ 'a\ iintervals) \subseteq Y$
using *assms* **by** *auto*

lemma *B03*:
$((X{::}\ 'a\ iintervals) = Y\ ) \longleftrightarrow (-X = -Y)$
by *auto*

lemma *B05*:
assumes $(X{::}\ 'a\ iintervals) \cup Y \subseteq Z$
shows $X \subseteq Z \wedge Y \subseteq Z$
using *assms* **by** *auto*

lemma *B06*:
assumes $X \subseteq Z \wedge Y \subseteq Z$
shows $(X{::}\ 'a\ iintervals) \cup Y \subseteq Z$
using *assms* **by** *auto*

lemma *B07*:
$(X{::}\ 'a\ iintervals) \cup Y \subseteq Z \longleftrightarrow X \subseteq Z \wedge Y \subseteq Z$
by *auto*

lemma *B08*:
assumes $(X{::}\ 'a\ iintervals) \subseteq Y$
shows $-X \cup Y = STrue$
using *assms*
using *strue-def* **by** *auto*

lemma *B10*:
$(X{::}\ 'a\ iintervals) \subseteq Y \longleftrightarrow -X \cup Y = STrue$
using *strue-def* **by** *auto*

lemma *B11*:
assumes $(X::\ 'a\ iintervals) \subseteq Y$
shows $X \cap -Y = SFalse$
using *assms sfalse-def* **by** *auto*

lemma *B12*:
assumes $X \cap -Y = SFalse$
shows $(X::\ 'a\ iintervals) \subseteq Y$
using *assms sfalse-def* **by** *auto*

lemma *B13*:
$(X::\ 'a\ iintervals) \subseteq Y \longleftrightarrow X \cap -Y = SFalse$
using *sfalse-def* **by** *auto*

lemma *B14*:
assumes $(X::\ 'a\ iintervals) \subseteq Y$
shows $X \cap Y = X$
using *assms* **by** *auto*

lemma *B15*:
assumes $(X::\ 'a\ iintervals) \subseteq Y \cap Z$
shows $X \subseteq Y \wedge X \subseteq Z$
using *assms* **by** *auto*

lemma *B16*:
assumes $X \subseteq Y \wedge X \subseteq Z$
shows $(X::\ 'a\ iintervals) \subseteq Y \cap Z$
using *assms* **by** *auto*

lemma *B17*:
$(X::\ 'a\ iintervals) \subseteq Y \cap Z \longleftrightarrow X \subseteq Y \wedge X \subseteq Z$
by *auto*

lemma *B18*:
assumes $(X::\ 'a\ iintervals) \subseteq Y \cup Z$
shows $X \cap -Y \subseteq Z$
using *assms* **by** *auto*

lemma *B19*:
assumes $X \cap -Y \subseteq Z$
shows $(X::\ 'a\ iintervals) \subseteq Y \cup Z$
using *assms* **by** *auto*

lemma *B21*:
$(X::\ 'a\ iintervals) \cup (Y \cap Z) \subseteq (X \cup Y) \cap (X \cup Z) \longleftrightarrow$
$X \cup (Y \cap Z) \subseteq (X \cup Y) \wedge X \cup (Y \cap Z) \subseteq (X \cup Z)$
by *auto*

lemma *B22*:

$(X{::}\ 'a\ iintervals) \cup (Y \cap Z) \subseteq X \cup Y$
by *auto*

lemma *B23*:
$(X{::}\ 'a\ iintervals) \cup (Y \cap Z) \subseteq X \cup Z$
by *auto*

lemma *B24*:
$((X{::}\ 'a\ iintervals) \cup Y) \cap (X \cup Z) \subseteq X \cup (Y \cap Z) \longleftrightarrow$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z$
by *auto*

lemma *B25*:
$(((X{::}\ 'a\ iintervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z \longleftrightarrow$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \wedge$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$
by *auto*

lemma *B26*:
$(((X{::}\ 'a\ iintervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y$
by *auto*

lemma *B27*:
$(((X{::}\ 'a\ iintervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$
by *auto*

lemma *B29*:
$(X{::}\ 'a\ iintervals) \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
by *auto*

Chop

lemma *CH02*:
$X{\cdot}Y \cap -(X{\cdot}Z) \subseteq X{\cdot}(Y \cap -Z)$
by (*metis B20 FusionRuleR FusionUnionDistR inf.right-idem inf-le1*)

lemma *CH03*:
$X{\cdot}Y \cap -(Z{\cdot}Y) \subseteq (X \cap -Z){\cdot}Y$
by (*metis B20 FusionRuleL FusionUnionDistL inf.right-idem inf-le1*)

lemma *CH04*:
$X{\cdot}Y \cap -(X{\cdot}-Z) \subseteq X{\cdot}(Y \cap Z)$
using *CH02* **by** *fastforce*

lemma *CH05*:
$X{\cdot}Y \cap -(-Z{\cdot}Y) \subseteq (X \cap Z){\cdot}Y$
using *CH03* **by** *fastforce*

lemma *CH06*:
assumes $X{\subseteq}X1$

$Y \subseteq Y1$
shows $X \cdot Y \subseteq X1 \cdot Y1$
using *assms*
by (*metis FusionRuleL FusionRuleR order-trans*)

lemma *CH11*:
$((X \cap (SPower\ SSkip\ n)) \cdot STrue) \cap ((SPower\ SSkip\ n) \cdot Y) = (X \cap (SPower\ SSkip\ n)) \cdot Y$
using *PwrFusionInterL*[*of n X STrue STrue Y*]
by (*simp add*: *inf-commute strue-def*)

lemma *CH12*:
$(STrue \cdot (X \cap (SPower\ SSkip\ n))) \cap (Y \cdot (SPower\ SSkip\ n)) = (Y \cdot (X \cap (SPower\ SSkip\ n)))$
using *PwrFusionInterR*[*of STrue n X Y STrue*]
by (*metis STrueTop inf-commute inf-sup-absorb*)

lemma *CH13*:
$(SPower\ SSkip\ n) \cdot (SPower\ SSkip\ m) = (SPower\ SSkip\ (n+m))$
proof
(*induct n arbitrary*: *m*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyL*)
next
case (*Suc n*)
then show *?case*
by (*metis FusionAssoc add-Suc pwr-Suc*)
qed

Next

lemma *N01*:
$(SNext\ SEmpty) = SSkip$
by (*simp add*: *FusionSEmptyR snext-def*)

lemma *N02*:
$(SNext\ SFalse) = SFalse$
by (*metis FusionSFalse SStar16 SStarSkip disjoint-eq-subset-Compl sfalse-def sfinite-def snext-def*)

lemma *N03*:
$(SNext\ X) \cdot Y = (SNext\ (X \cdot Y))$
by (*simp add*: *snext-def FusionAssoc*)

lemma *N04*:
$(SNext\ (X \cup Y)) = (SNext\ X) \cup (SNext\ Y)$
by (*simp add*: *FusionUnionDistR snext-def*)

lemma *N05*:
$(SNext\ (X \cap Y)) = (SNext\ X) \cap (SNext\ Y)$
by (*metis CH07 SStar16 inf.orderE snext-def*)

lemma *N06*:

assumes $X \subseteq Y$
shows $(SNext\ X) \subseteq (SNext\ Y)$
using *assms*
by (*metis FusionUnionDistR Subsumption snext-def*)

lemma *N07*:
$(SNext\ ((-X) \cup Y)) = (SNext\ (-X)) \cup (SNext\ Y)$
by (*simp add*: *N04*)

lemma *N08*:
$SMore \subseteq SSkip \cdot STrue$
using *SMoreImpSSkipFusion* **by** *blast*

lemma *N23*:
$(SWprev\ X) \subseteq (SEmpty \cup (SPrev\ X))$
proof −
have *1*: $(X \cap SFinite) \cdot SSkip \cup (-X \cap SFinite) \cdot SSkip = SStar\ SSkip \cdot SSkip$
by (*metis B28 FusionUnionDistL SStarSkip STrueTop boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)
have *2*: $(SEmpty) \cup STrue \cdot SSkip = STrue$
by (*metis FusionSFalse FusionUnionDistL Powerstarhelp2 SStar19 SStarSkip UnfoldL UnionAssoc compl-bot-eq compl-sup-top sfinite-def sinf-def strue-def*)
have *3*: $-SWprev\ X \cup (SEmpty \cup SPrev\ X) = STrue$
unfolding *swprev-def sprev-def*
by (*metis 2 FusionUnionDistL compl-bot-eq compl-sup-top double-compl inf-sup-aci*(*7*) *strue-def*)
then show *?thesis* **by** (*meson B09*)
qed

lemma *N24*:
$(SEmpty) \subseteq (SWprev\ X)$
proof −
have *1*: $(-SEmpty \cap -(-SEmpty \cdot -SEmpty)) \subseteq SMore$
by (*simp add*: *smore-def*)
have *2*: $-X \cdot SMore \subseteq SMore$
by (*metis CH01 FusionAssoc1 FusionUnionDistL SMoreImpSSkipFusion SSkipFusionImpSMore SStar30 SStar48 STrueTop subset-antisym sup.orderI sup.right-idem*)
have *3*: $-X \cdot (-SEmpty \cap -(-SEmpty \cdot -SEmpty)) \subseteq SMore$
using *1 2 FusionRuleR* **by** *blast*
show *?thesis*
by (*metis 3 compl-le-swap1 compl-sup smore-def sskip-def swprev-def*)
qed

lemma *N25*:
$(SPrev\ X) \subseteq (SWprev\ X)$
proof −
have *1*: $((SPrev\ X) \subseteq (SWprev\ X)) = (((SPrev\ X) \cap (SPrev\ (-X))) \subseteq SFalse)$
by (*simp add*: *B10 sfalse-def sprev-def swprev-def*)
have *2*: $((SPrev\ X) \cap (SPrev\ (-X))) = (X \cap (-X)) \cdot SSkip$
by (*metis CH08 SStar16 inf.orderE sprev-def*)
have *3*: $(X \cap (-X)) \cdot SSkip = SFalse \cdot SSkip$

by (*simp add*: *sfalse-def*)
have *4*: *SFalse·SSkip* = *SFalse*
by (*simp add*: *SFalseFusion*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *N26*:
(*SWprev X*) = (*SEmpty* ∪ (*SPrev X*))
using *N23 N24 N25* **by** *blast*

lemma *N09*:
SSkip ∪ *SMore·SSkip* ⊆ *SMore*
proof −
have *1*: *SSkip* ⊆ *SMore*
by (*simp add*: *smore-def sskip-def*)
have *2*: *SMore·SSkip* ⊆ *SMore*
by (*metis N26 UnionCommute compl-le-swap1 smore-def sup-ge2 swprev-def*)
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *N10*:
assumes *SSkip* ∪ *SMore·SSkip* ⊆ *SMore*
shows *SSkip·*(*SStar SSkip*) ⊆ *SMore*
using *assms SStarInductR N09* **by** *blast*

lemma *N11*:
SSkip · *STrue* ⊆ *SMore*
by (*simp add*: *SSkipFusionImpSMore*)

lemma *N12*:
(*SNext X*) = −(*SWnext* (−*X*))
by (*simp add*: *snext-def swnext-def*)

lemma *N13*:
SMore·STrue = *SMore*
by (*metis CH01 FusionAssoc1 SMoreImpSSkipFusion SSkipFusionImpSMore subset-antisym*)

lemma *N14*:
STrue·SSkip ⊆ *SMore*
by (*metis N09 ST47 STrueTop smore-def*)

lemma *N15*:
SMore ⊆ *STrue·SSkip*
proof −
have *1*: *SMore* ⊆ *SSkip* · *STrue*
by (*simp add*: *SMoreImpSSkipFusion*)
have *2*: *SSkip* · *STrue* ⊆ *STrue·SSkip*
proof −
have *f3*: *SSkip* · *SFinite* ⊆ *STrue* · *SSkip*
by (*metis B19 FusionRuleL SStar19 SStarSkip STrueTop inf-sup-ord*(*2*))

```
    have SSkip · SInf ⊆ STrue · SSkip
      by (metis (no-types, opaque-lifting) B04 Compl-disjoint2 FusionRuleR SChopAssoc SMoreImpSSkip-
Fusion
          SSkipFusionImpSMore ST47 boolean-algebra-cancel.inf0 compl-inf double-compl
          inf-commute inf-le1 sfalse-def sinf-def strue-def)
    then show ?thesis
      using f3 by (metis (no-types) FusionUnionDistR STrueTop le-sup-iff sfinite-def)
  qed
 show ?thesis
   using 2 SMoreImpSSkipFusion by blast
qed

lemma N19:
 (SWnext X) ⊆ (SEmpty ∪ (SNext X))
proof −
  have STrue ⊆ SSkip · (− X) ∪ (SEmpty ∪ SSkip · X)
    by (metis B04 FusionUnionDistR N13 SMoreImpSSkipFusion SSkipFusionImpSMore SStar48 UnfoldL
        compl-bot-eq compl-sup-top inf-sup-aci(7) strue-def)
  then show ?thesis
    by (simp add: B13 snext-def strue-def swnext-def)
qed

lemma N20:
  (SEmpty ) ⊆ (SWnext X)
proof −
 have 1: ((SEmpty ) ⊆ (SWnext X) ) = (−(SWnext X) ⊆ SMore)
   by (simp add: smore-def)
 have 2: (−(SWnext X) ⊆ SMore) = ((SNext (−X)) ⊆ SMore)
   by (simp add: snext-def swnext-def)
 have 3: (SNext (−X)) ⊆ SSkip·STrue
   by (metis FusionUnionDistR STrueTop snext-def sup.orderI sup.right-idem)
 hence 4: (SNext (−X)) ⊆ SMore  using SSkipFusionImpSMore by auto
 from 1 2 4 show ?thesis by auto
qed

lemma N21:
  (SEmpty ∪ (SNext X)) ⊆ (SWnext X)
using N20
by (metis B06 SNextImpNotNextNot snext-def swnext-def)

lemma N22:
 (SWnext X) = (SEmpty ∪ (SNext X))
using N21 N19 by blast

lemma N16:
  (SNext X) = SMore ∩ (SWnext X)
using N12 N22 smore-def by blast

lemma N17:
  (SWnext (X ∩ Y)) = (SWnext X) ∩ (SWnext Y)
```

by (*simp add*: *N05 N22 Un-Int-distrib*)

lemma *N18*:
(*SWnext* (*X* ∪ *Y*)) = (*SWnext X*) ∪ (*SWnext Y*)
by (*simp add*: *swnext-def*)
(*metis CH07 SSkipSFinite compl-inf*)

lemma *N27*:
(*SNext* ((−*X*) ∪ *Y*)) ⊆ (−(*SNext X*) ∪ (*SNext Y*))
using *N04 SNextImpNotNextNot* **by** *blast*

lemma *N28*:
(*SPrev* ((−*X*) ∪ *Y*)) ⊆ (−(*SPrev X*) ∪ (*SPrev Y*))
unfolding *sprev-def*
proof −
have ⋀*I*. (*SEmpty*) ∪ *I* · *SSkip* = − (− *I* · *SSkip*)
using *N26 sprev-def swprev-def* **by** *blast*
then have (*Y* ∪ − *X*) · *SSkip* ⊆ *Y* · *SSkip* ∪ − (*X* · *SSkip*)
using *FusionUnionDistL* **by** *blast*
then show (− *X* ∪ *Y*) · *SSkip* ⊆ − (*X* · *SSkip*) ∪ *Y* · *SSkip*
by (*simp add*: *UnionCommute*)
qed

lemma *N29*:
(*SPrev X*) = −(*SWprev* (−*X*))
by (*simp add*: *sprev-def swprev-def*)

SInit

lemma *ST01*:
(*X* ∩ *SEmpty*)·*Y* ⊆ *Y*
by (*metis Int-commute FusionUnionDistL FusionSEmptyL le-iff-sup sup-inf-absorb UnionCommute*)

lemma *ST02*:
(*X* ∩ *SEmpty*)·*Y* ⊆ (*X* ∩ *SEmpty*)·*STrue*
by (*simp add*: *FusionRuleR strue-def*)

lemma *ST03*:
(*X* ∩ *SEmpty*)·(*X* ∩ *SEmpty*) ⊆ (*X* ∩ *SEmpty*)
using *ST01* **by** *auto*

lemma *ST04*:
(*X* ∩ *SEmpty*) ⊆ (*X* ∩ *SEmpty*)·(*X* ∩ *SEmpty*)
proof −
have ∀ *S Sa Sb Sc*. (*Sc*) · (*Sb* ∩ *SEmpty*) ∩ (*Sa* · (*S* ∩ *SEmpty*)) = *Sc* ∩ *Sa* · (*Sb* ∩ (*S* ∩ *SEmpty*))
by (*simp add*: *CH10 inf-assoc*)
then have ∀ *S Sa*. (*Sa*) · (*S* ∩ *SEmpty*) · (*S* ∩ *SEmpty*) = *Sa* · (*S* ∩ *SEmpty*)
by (*metis FusionSEmptyR inf.idem inf-commute*)
then show *?thesis*
by (*metis FusionSEmptyL subset-refl*)

qed

lemma *ST05*:
$(X \cap SEmpty) \subseteq -((-X) \cap SEmpty)$
by *blast*

lemma *ST06*:
$((-X) \cap SEmpty) \subseteq -(X \cap SEmpty)$
by *auto*

lemma *ST07*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq (X \cap SEmpty) \cdot STrue$
using *ST02 FusionSEmptyR* **by** *blast*

lemma *ST08*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq (STrue \cap SEmpty) \cdot (Y \cap SEmpty)$
by (*metis FusionSEmptyL FusionSEmptyR ST33 inf.cobounded2*)

lemma *ST09*:
$((X \cap SEmpty) \cdot STrue) \cap (STrue \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (X \cap SEmpty) \cdot (Y \cap SEmpty)$
by (*metis compl-bot-eq eq-refl FusionSEmptyR inf.commute inf-top.left-neutral CH09 strue-def*)

lemma *ST10*:
$(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (X \cap SEmpty)$
by (*metis FusionRuleR FusionSEmptyR inf-le2*)

lemma *ST11*:
$(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (Y \cap SEmpty)$
using *ST01* **by** *blast*

lemma *ST12*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) = (X \cap SEmpty) \cdot SEmpty \cap (Y \cap SEmpty) \cdot SEmpty$
by (*simp add*: *FusionSEmptyR*)

lemma *ST14*:
$((X \cap Y) \cap SEmpty) \cdot SEmpty = ((X \cap Y) \cap SEmpty)$
by (*simp add*: *FusionSEmptyR*)

lemma *ST16*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq SEmpty$
by (*simp add*: *le-infI2*)

lemma *ST17*:
$(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq SEmpty$
using *ST10* **by** *auto*

lemma *ST18*:
$-((X \cap SEmpty) \cup (Y \cap SEmpty)) = -(X \cap SEmpty) \cap -(Y \cap SEmpty)$
by *auto*

lemma *ST19*:
$(X \cap SEmpty)\cdot((-X) \cap SEmpty) \subseteq (X \cap SEmpty)$
using *ST10* **by** *blast*

lemma *ST20*:
$(X \cap SEmpty)\cdot((-X) \cap SEmpty) \subseteq ((-X) \cap SEmpty)$
using *ST01* **by** *auto*

lemma *ST22*:
$((X \cap SEmpty)\cdot SSkip)\cdot(Y \cap SEmpty) \subseteq (X \cap SEmpty)\cdot SSkip$
using *FusionRuleR FusionSEmptyR* **by** *blast*

lemma *ST23*:
$((X \cap SEmpty)\cdot SSkip)\cdot(Y \cap SEmpty) \subseteq SSkip\cdot(Y \cap SEmpty)$
by (*simp add*: *ST01 FusionRuleL*)

lemma *ST27*:
$(SInit\ X) \cap (Y\cdot Z) \subseteq ((SInit\ X) \cap Y)\cdot Z$
by (*metis B04 compl-bot-eq FusionAssoc FusionSEmptyL inf-commute inf-top.left-neutral*
CH09 sinit-def strue-def)

lemma *ST29*:
$(SInit\ X)\cdot Y \subseteq (SInit\ X)$
using *ST02 FusionAssoc sinit-def* **by** *fastforce*

lemma *ST30*:
$(SInit\ X) \cap (SDi\ Y) = (SDi\ (\ (SInit\ X) \cap Y))$
unfolding *sdi-def sinit-def strue-def*
by (*metis CH09 FusionAssoc FusionSEmptyL compl-bot-eq inf-top.left-neutral*)

lemma *ST31*:
$(X\cdot(STrue \cap SEmpty)) \cap (STrue\cdot(Y \cap SEmpty)) = X\cdot(Y \cap SEmpty)$
by (*metis Int-commute compl-bot-eq inf-top.right-neutral CH10 strue-def*)

lemma *ST32*:
$(STrue \cap SEmpty)\cdot SEmpty \cap (SInit\ X) = (X \cap SEmpty)$
proof −
have $\forall S\ Sa.\ (Sa) \cap (S \cap Sa) = S \cap Sa$
by *fastforce*
then have *f1*: $\forall S\ Sa.\ (Sa) \cap (S \cap SEmpty) \subseteq Sa \cap SEmpty \cdot STrue$
by (*metis* (*no-types*) *ST07 inf-assoc*)
have *f4*: $\forall S.\ -\ (-\ (S::'a\ iintervals)) = S$
by *blast*
have *f6*: $\forall S.\ (SEmpty) \cap (S \cap (S \cap SEmpty \cdot STrue)) = S \cap SEmpty$
using *f1* **by** *auto*
have *f7*: $\forall S.\ (SEmpty) \cap (SEmpty \cap S \cdot STrue) \subseteq S$
using *f4* **by** (*metis CH11 FusionSEmptyR inf-aci*(*1*) *inf-le2 pwr-0*)
have $\forall S.\ (SEmpty) \cap (S \cap (SEmpty \cap S \cdot STrue)) = SEmpty \cap S$
using *f6* **by** (*simp add*: *inf-commute*)
then have $SEmpty \cap (SEmpty \cap X \cdot STrue) = SEmpty \cap X$

using *f7* **by** *auto*
then show *?thesis*
by (*metis* (*no-types*) *ST33 inf-commute sinit-def*)
qed

lemma *ST34*:
$((X \cap SEmpty)\cdot Y) = (SInit\ X) \cap Y$
by (*metis FusionSEmptyL Int-commute CH09 compl-bot-eq inf-top-right sinit-def strue-def*)

lemma *ST35*:
$((SInit\ X) \cap Y)\cdot Z \subseteq (SInit\ X) \cap (Y\cdot Z)$
by (*metis B04 ST34 FusionAssoc*)

lemma *ST39*:
$SEmpty \cap (SInit\ X) \subseteq (X \cap SEmpty)$
using *ST32* **by** *blast*

lemma *ST40*:
$(X \cap SEmpty) \subseteq SEmpty \cap (SInit\ X)$
using *ST32* **by** *auto*

lemma *ST41*:
$SEmpty \cap (SInit\ X) = (X \cap SEmpty)$
using *ST40 ST39* **by** *auto*

lemma *ST42*:
$(X \cap SEmpty) \subseteq ((X \cup Y) \cap SEmpty)$
by *blast*

lemma *ST43*:
$(Y \cap SEmpty) \subseteq ((X \cup Y) \cap SEmpty)$
by *blast*

lemma *ST44*:
$(X \cap SEmpty) \cap ((-X) \cap SEmpty) = SFalse$
by (*simp add*: *sfalse-def*)

lemma *ST45*:
$((X \cup Y) \cap SEmpty) \subseteq (X \cap SEmpty) \cup (Y \cap SEmpty)$
by *auto*

lemma *ST46*:
$(SInit\ X) \cup (SInit\ Y) = (SInit\ (X \cup Y))$
by (*simp add*: *Int-Un-distrib2 FusionUnionDistL sinit-def*)

lemma *ST48*:
$-(STrue\cdot(X \cap SEmpty)) \subseteq STrue\cdot((-X) \cap SEmpty)$
by (*metis B09 FusionSEmptyR FusionUnionDistR ST21 double-compl*)

SStar

lemma *SStar02*:
assumes $X \subseteq Y$
shows $X{\cdot}(SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$
using *assms UnfoldL*[*of Y*]
by (*metis B05 B06 FusionRuleL Subsumption sup.cobounded2*)

lemma *SStar04*:
$(SStar\ X) \subseteq (SStar\ X){\cdot}(SStar\ X)$
by (*metis Un-absorb UnfoldL UnionAssoc ST47 sup.absorb-iff2*)

lemma *SStar09*:
assumes $(X{\cdot}(SEmpty \cup (X{\cdot}(SStar\ X)))) \cup SEmpty \subseteq (SEmpty \cup (X{\cdot}(SStar\ X)))$
shows $(SStar\ X) \subseteq SEmpty \cup (X{\cdot}(SStar\ X))$
using *assms*
by (*simp add*: *UnfoldL*)

lemma *SStar10*:
$(X{\cdot}(SEmpty \cup (X{\cdot}(SStar\ X)))) \subseteq (SEmpty \cup (X{\cdot}(SStar\ X)))$
by (*metis UnfoldL sup-ge2*)

lemma *SStar11*:
$SEmpty \subseteq (SEmpty \cup (X{\cdot}(SStar\ X)))$
by *auto*

lemma *SStar13*:
$(SStar\ SSkip) = SFinite$
by (*simp add*: *SStarSkip*)

lemma *SStar14*:
$(SSometime\ X) = (SStar\ SSkip){\cdot}X$
by (*simp add*: *SStarSkip ssometime-def*)

lemma *SStar20*:
$(SStar\ SEmpty) = SEmpty$
by (*metis FusionSEmptyR ST15 ST33*)

lemma *SStar21*:
$(SStar\ (SEmpty \cap X)){\cdot}(SEmpty \cap X) = (SEmpty \cap X)$
by (*metis ST15 FusionSEmptyL inf-commute*)

lemma *SStar24*:
$(SStar\ SFalse) = SEmpty$
by (*metis SStar20 SStar47 inf-compl-bot sfalse-def smore-def*)

lemma *SStar26*:
$X \subseteq (SStar\ X)$
using *UnfoldL*[*of X*]
by (*metis B05 FusionSEmptyR FusionUnionDistR SStar10*)

lemma *SStar27*:
SEmpty $\subseteq$ (*SStar X*)
using *UnfoldL* **by** *blast*

lemma *SStar31*:
assumes $X \cup (X \cdot Y) \cdot (X \cdot (SStar\ (Y \cdot X))) \subseteq X \cdot (SStar\ (Y \cdot X))$
shows $(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$
using *assms SStarInductL* **by** *blast*

lemma *SStar32*:
$X \cup (X \cdot Y) \cdot (X \cdot (SStar\ (Y \cdot X))) \subseteq X \cdot (SStar\ (Y \cdot X))$
by (*metis B06 SStar10 SStar11 FusionAssoc FusionRuleR FusionSEmptyR UnfoldL*)

lemma *SStar33*:
$(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$
by (*meson SStar32 SStarInductL*)

lemma *SStar37*:
assumes $X \cdot Z \subseteq Z \cdot Y$
shows $(SStar\ X) \cdot Z \subseteq Z \cdot (SStar\ Y)$
proof −
have $Z \cdot SStar\ Y = Z \cdot SEmpty \cup Z \cdot (Y \cdot SStar\ Y)$
by (*metis FusionUnionDistR UnfoldL*)
then have $Z \cdot SStar\ Y \cup (Z \cup X \cdot (Z \cdot SStar\ Y)) = Z \cup Z \cdot Y \cdot SStar\ Y \cup X \cdot Z \cdot SStar\ Y$
using *FusionAssoc FusionSEmptyR* **by** *blast*
then have $Z \cdot SStar\ Y \cup (Z \cup X \cdot (Z \cdot SStar\ Y)) = Z \cdot SStar\ Y$
by (*metis* (*no-types*) *FusionAssoc FusionSEmptyR FusionUnionDistL FusionUnionDistR UnfoldL UnionAssoc*
assms sup.absorb-iff1)
then show *?thesis*
by (*meson SStarInductL sup.absorb-iff1*)
qed

lemma *SStar38*:
assumes $Z \cdot X \subseteq Y \cdot Z$
shows $Z \cdot (SStar\ X) \subseteq (SStar\ Y) \cdot Z$
using *assms*
proof −
have *f1*: $Z \cup SStar\ Y \cdot Y \cdot Z = SStar\ Y \cdot Z$
by (*metis* (*no-types*) *SStar30 ST47 UnfoldL*)
have $SStar\ Y \cdot Y \cdot Z = SStar\ Y \cdot Z \cdot X \cup SStar\ Y \cdot Y \cdot Z$
by (*metis FusionAssoc FusionUnionDistR assms subset-Un-eq*)
then have $Z \cup SStar\ Y \cdot Z \cdot X \subseteq SStar\ Y \cdot Z$
using *f1* **by** *blast*
then show *?thesis*
by (*simp add*: *SStarInductR*)
qed

lemma *SStar39*:
$Y \cdot (SStar\ ((SStar\ X) \cdot Y)) \subseteq (SStar\ (Y \cdot (SStar\ X))) \cdot Y$

by (*simp add*: *SStar38 FusionAssoc*)

lemma *SStar40*:
(*SStar* (*Y*·(*SStar X*)))·*Y* ⊆ *Y*·(*SStar* ((*SStar X*)·*Y*))
by (*simp add*: *SStar33*)

lemma *SStar41*:
Y·(*SStar* ((*SStar X*)·*Y*)) = (*SStar* (*Y*·(*SStar X*)))·*Y*
using *SStar39 SStar40* **by** *blast*

lemma *SStar42*:
Z·(*SStar* (*Y*·*Z*)) ⊆ (*SStar* (*Z*·*Y*))·*Z*
by (*simp add*: *SStar38 FusionAssoc*)

lemma *SStar43*:
(*SStar* (*Z*·*Y*))·*Z* ⊆ *Z*·(*SStar* (*Y*·*Z*))
by (*simp add*: *SStar33*)

lemma *SStar44*:
Z·(*SStar* (*Y*·*Z*)) = (*SStar* (*Z*·*Y*))·*Z*
using *SStar42 SStar43* **by** *blast*

lemma *SStar49*:
(*SStar X*) = *SEmpty* ∪ (*SStar X*)·*X*
using *SStar30 UnfoldL* **by** *blast*

Box and Diamond

lemma *BD01*:
(*SSometime SEmpty*) = *SFinite*
by (*simp add*: *ssometime-def FusionSEmptyR*)

lemma *BD02*:
X ⊆ (*SSometime X*)
unfolding *ssometime-def*
by (*metis SStar15 SStarSkip ST47 subset-Un-eq*)

lemma *BD03*:
(*SNext* (*SSometime X*)) ⊆ (*SSometime X*)
by (*metis FusionUnionDistL N03 SStar16 SStar03 SStar04 SStarSkip snext-def ssometime-def*
sup.absorb-iff2 sup.orderE)
lemma *BD04*:
(*SSometime* (*SNext X*)) ⊆ (*SSometime X*)
by (*metis BD03 FusionAssoc SStar28 SStar29 SStarSkip inf-sup-aci*(5) *snext-def ssometime-def sup.orderE*)

lemma *BD05*:
(*SSometime X*) ∪ (*SSometime Y*) = (*SSometime* (*X* ∪ *Y*))
by (*simp add*: *FusionUnionDistR ssometime-def*)

lemma *BD06*:

$(SSometime\ STrue) = STrue$
using *BD02 SMP* **by** *blast*

lemma *BD07*:
$(SSometime\ (X \cap Y)) \subseteq (SSometime\ X) \cap (SSometime\ Y)$
by (*simp add*: *FusionRuleR ssometime-def*)

lemma *BD08*:
$(SAlways\ STrue) = STrue$
by (*simp add*: *SBoxGen*)

lemma *BD09*:
$-(SAlways\ X) = (SSometime\ (-X))$
by (*simp add*: *salways-def*)

lemma *BD10*:
$(SAlways\ X) \subseteq (SSometime\ X)$
by (*metis B02 BD02 BD09 set-rev-mp subsetI*)

lemma *BD11*:
$(SSometime\ (SSometime\ X)) = (SSometime\ X)$
by (*metis FusionAssoc SStar03 SStar04 SStarSkip ssometime-def subset-antisym*)

lemma *BD12*:
$(SAlways\ X) \subseteq X$
by (*simp add*: *B02 BD02 BD09*)

lemma *BD13*:
$(SDi\ STrue) = STrue$
by (*simp add*: *CH01 sdi-def*)

lemma *BD14*:
$(SDi\ SEmpty) = STrue$
by (*simp add*: *sdi-def FusionSEmptyL*)

lemma *BD15*:
$(SBi\ STrue) = STrue$
by (*simp add*: *SBiGen*)

lemma *BD16*:
$(SDi\ (X \cup Y)) = (SDi\ X) \cup (SDi\ Y)$
by (*simp add*: *FusionUnionDistL sdi-def*)

lemma *BD17*:
assumes $X \subseteq Y$
shows $(SDi\ X) \subseteq (SDi\ Y)$
using *assms*
by (*metis FusionUnionDistL Subsumption sdi-def*)

lemma *BD18*:

$(SDi\ (SDi\ X)) = (SDi\ X)$
by (*metis CH01 FusionAssoc sdi-def*)

lemma *BD19*:
$(SDa\ SEmpty) = STrue$
by (*metis BD01 BD06 sda-def ssometime-def*)

lemma *BD20*:
$(SDa\ STrue) = STrue$
by (*metis BD06 CH01 sda-def ssometime-def*)

lemma *BD21*:
$(SBa\ STrue) = STrue$
by (*metis BD15 BD08 BD09 sba-def sbi-def sda-def sdi-def ssometime-def*)

lemma *BD22*:
$(SDa\ (X \cup Y)) = (SDa\ X) \cup (SDa\ Y)$
by (*simp add*: *FusionUnionDistL FusionUnionDistR sda-def*)

lemma *BD23*:
assumes $X \subseteq Y$
shows $(SDa\ X) \subseteq (SDa\ Y)$
using *assms*
by (*metis BD22 Subsumption*)

lemma *BD24*:
assumes $X \subseteq Y$
shows $(SDa\ (-Y)) \subseteq (SDa\ (-X))$
using *assms*
by (*simp add*: *BD23*)

lemma *BD25*:
$(SDi\ X) \subseteq (SDa\ X)$
by (*metis BD02 FusionAssoc sda-def sdi-def ssometime-def*)

lemma *BD26*:
$(SSometime\ X) \subseteq (SDa\ X)$
by (*metis B08 B12 FusionRuleR FusionSEmptyR SFalseBottom double-compl inf.absorb-iff2 inf-le1 sda-def ssometime-def sup-inf-absorb*)

lemma *BD27*:
$(SBa\ X) \subseteq (SBi\ X)$
by (*simp add*: *BD25 sba-def sbi-def*)

lemma *BD28*:
$(SBa\ X) \subseteq (SAlways\ X)$
by (*simp add*: *B02 BD26 BD09 sba-def*)

lemma *BD29*:
$(SAlways\ X) \cap (SAlways\ Y) = (SAlways\ (X \cap Y))$

proof −
have *1*: (*SAlways X*) ∩ (*SAlways Y*) = − *SSometime* (− *X*) ∩ − *SSometime* (− *Y*)
by (*simp add* : *salways-def*)
have *2*: *SSometime* (− *X*) ∪ *SSometime* (− *Y*) = *SSometime*(−*X* ∪ −*Y*)
using *BD05* **by** *blast*
have *3*: − *SSometime*(−*X* ∪ −*Y*) = (*SAlways* (*X* ∩ *Y*))
by (*simp add* : *salways-def*)
show *?thesis* **using** *1 2 3* **by** *auto*
qed

lemma *BD30*:
(*SAlways X*) ∪ (*SAlways Y*) ⊆ (*SAlways* (*X* ∪ *Y*))
using *BD07*
by (*metis B02 BD09 compl-sup*)

lemma *BD31*:
(*SDi* (*X* ∩ *Y*)) ⊆ (*SDi X*) ∩ (*SDi Y*)
by (*simp add*: *BD17*)

lemma *BD32*:
(*SBi X*) ∪ (*SBi Y*) ⊆ (*SBi* (*X* ∪ *Y*))
using *BD31*
by (*metis* (*mono-tags*, *lifting*) *B02 compl-sup double-compl sbi-def*)

lemma *BD33*:
(*SDa* (*X* ∩ *Y*)) ⊆ (*SDa X*) ∩ (*SDa Y*)
by (*simp add*: *BD23*)

lemma *BD34*:
(*SBa X*) ∪ (*SBa Y*) ⊆ (*SBa* (*X* ∪ *Y*))
using *BD33*
by (*metis* (*mono-tags*, *lifting*) *B02 compl-sup double-compl sba-def*)

lemma *BD35*:
(*SAlways SEmpty*) = *SEmpty*
by (*metis B08 BD08 BD12 FusionSEmptyR N20 SBoxInduct ST33 sup.absorb-iff2 sup.orderE*)

lemma *BD36*:
(*SBi SEmpty*) = *SEmpty*
proof −
have *1*: − (− *SEmpty* · *STrue*) ⊆ *SEmpty*
by (*metis B04 N13 double-compl smore-def*)
have *2*: *SEmpty* ⊆ − (− *SEmpty* · *STrue*)
using *N13 smore-def* **by** *fastforce*
show *?thesis*
by (*simp add*: *1 2 B04 sbi-def sdi-def*)
qed

lemma *BD37*:
(*SBa SEmpty*) = *SEmpty*

by (*metis BD09 BD35 BD36 sba-def sbi-def sda-def sdi-def ssometime-def*)

lemma *BD38*:
assumes $X \subseteq Y$
shows $(SAlways\ X) \subseteq (SAlways\ Y)$
using *assms*
by (*simp add*: *BD29 inf.absorb-iff2*)

lemma *BD39*:
assumes $X \subseteq Y$
shows $(SBi\ X) \subseteq (SBi\ Y)$
using *assms*
by (*simp add*: *BD17 sbi-def*)

lemma *BD40*:
assumes $X \subseteq Y$
shows $(SBa\ X) \subseteq (SBa\ Y)$
using *assms*
by (*simp add*: *BD24 sba-def*)

lemma *BD41*:
$(SBi\ (SBi\ X)) = (SBi\ X)$
by (*simp add*: *BD18 sbi-def*)

lemma *BD42*:
$(SAlways\ (SAlways\ X)) = (SAlways\ X)$
by (*simp add*: *BD11 salways-def*)

lemma *BD43*:
$(SDa\ (SDa\ X)) = (SDa\ X)$
by (*metis BD11 CH01 FusionAssoc sda-def ssometime-def*)

lemma *BD44*:
$(SBa\ (SBa\ X)) = (SBa\ X)$
by (*simp add*: *BD43 sba-def*)

lemma *BD47*:
$(SAlways\ (\ (-X) \cup Y)) \subseteq (\ -(SAlways\ X) \cup (SAlways\ Y))$
by (*metis B20 BD12 BD29 BD38 BD42 double-compl*)

lemma *BD48*:
$(SAlways\ X) \subseteq X \cap (SWnext\ (SAlways\ X))$
by (*metis B02 B16 BD03 BD09 BD12 N12 salways-def*)

lemma *BD49*:
$(SBi\ (\ (-X) \cup Y)) \subseteq (-(SBi\ X) \cup (SBi\ Y)\ )$
by (*metis B20 BD45 Un-commute double-complement sbi-def sdi-def*)

lemma *BD50*:
$(SPrev\ (SDi\ X)) \subseteq (SDi\ X)$

by (*simp add*: *FusionAssoc1 FusionRuleR sdi-def sprev-def strue-elim subsetI*)

lemma *BD51*:
$-(SBi\ X) = (SDi\ (-X))$
by (*simp add*: *sbi-def*)

lemma *BD52*:
$X \subseteq (SDi\ X)$
by (*metis FusionSEmptyR FusionUnionDistR ST33 Subsumption UnionCommute sdi-def sup-inf-absorb*)

lemma *BD53*:
$(SBi\ X) \subseteq X$
by (*simp add*: *B02 BD51 BD52*)

lemma *BD54*:
$(SBi\ X) \subseteq X \cap (SWprev\ (SBi\ X))$
by (*metis B02 B16 BD50 BD51 BD53 N29 sbi-def*)

lemma *BD55*:
$(SBi\ (SMore \cup X)) = (SInit\ X)$
by (*metis* (*no-types*, *lifting*) *ST38 compl-sup double-complement inf-commute sbi-def sdi-def sinit-def smore-def*)

lemma *BD56*:
$(SAlways\ (SMore \cup X)) = STrue{\cdot}(X \cap SEmpty)$
proof −
have *1*: $((SFinite{\cdot}(X \cap SEmpty)) \cup SInf) = (STrue{\cdot}(X \cap SEmpty))$
by (*metis Powerstarhelp2 Powerstarhelp3 UnionCommute boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)
have *11*: $SInf \cup SFinite \cdot (SEmpty \cap X \cup SEmpty \cap - X) = STrue$
by (*simp add*: *B28 FusionSEmptyR sfinite-def strue-def*)
have *2*: $(SAlways\ (SMore \cup X)) \cap SFinite \subseteq SFinite{\cdot}(X \cap SEmpty)$
using *11*
by (*simp add*: *B09 BD09 FusionUnionDistR UnionCommute inf-commute inf-sup-aci*(*7*) *smore-def ssometime-def sfinite-def*)
have *3*: $(SAlways\ (SMore \cup X)) \subseteq ((SFinite{\cdot}(X \cap SEmpty)) \cup SInf)$
using *2 sfinite-def* **by** *fastforce*
have *4*: $(SAlways\ (SMore \cup X)) \subseteq\ STrue{\cdot}(X \cap SEmpty)$
using *1 3* **by** *blast*
have *5*: $SFinite{\cdot}(X \cap SEmpty) \subseteq (SAlways\ (SMore \cup X))$
unfolding *salways-def ssometime-def smore-def*
using *CH10*[*of SFinite X SFinite*] *FusionSFalse*
by (*metis* (*no-types*, *opaque-lifting*) *SFalseBottom compl-sup disjoint-eq-subset-Compl double-compl inf-commute inf-le2 sfinite-def*)
have *6*: $SInf \subseteq (SAlways\ (SMore \cup X))$
by (*metis B05 BD30 FusionSEmptyR double-compl salways-def sfinite-def smore-def ssometime-def*)
show *?thesis*
using *1 3 5 6* **by** *auto*
qed

lemma *BD57*:
(*SAlways H*) ⊆ *SAlways* (−*G* ∪ ((*SAlways H*)∩ *G*))
proof −
have *1*: (*SAlways H*) ⊆ (−*G* ∪ ((*SAlways H*)∩ *G*))
by *blast*
have *2*: *SAlways* (*SAlways H*) ⊆ *SAlways* (−*G* ∪ ((*SAlways H*)∩ *G*))
by (*simp add*: *1 BD38*)
have *3*: *SAlways* (*SAlways H*) = (*SAlways H*)
by (*simp add*: *BD42*)
show *?thesis* **using** *2 3* **by** *blast*
qed

lemma *BD58*:
((*SAlways H*) ∩ (*F*·*G*)) ⊆ (*F*·((*SAlways H*) ∩ *G*))
proof −
have *1*: *SAlways* (−*G* ∪ ((*SAlways H*)∩ *G*)) ∩ (*F*·*G*) ⊆ (*F*·((*SAlways H*) ∩ *G*))
using *BD46* **by** *blast*
show *?thesis* **using** *1 BD57* **by** *blast*
qed

Finite and Infinite

lemma *FI01*:
SFinite· *SFinite* = *SFinite*
by (*metis BD01 BD11 ssometime-def*)

lemma *FI02*:
(*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*) ⊆ (*X*·*Y*) ∩ *SFinite*
by (*metis B16 CH06 FI01 inf.cobounded1 inf-le2*)

lemma *FI03*:
(*X*·*Y*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
proof −
have *1*: (*X*·*Y*) = (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*) ∪ (*X* ∩ *SFinite*)·(*Y* ∩ *SInf*)
∪ (*X* ∩ *SInf*)·(*Y* ∩ *SFinite*) ∪ (*X* ∩ *SInf*)·(*Y* ∩ *SInf*)
by (*metis FusionUnionDistR Powerstarhelp2 Powerstarhelp3 SInfSFinite UnionCommute sup.right-idem*)
have *2*: ((*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
by *auto*
have *3*: (*X* ∩ *SFinite*)·(*Y* ∩ *SInf*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
by (*metis* (*no-types*, *lifting*) *B20 FusionAssoc FusionSFalse inf-le2 sfinite-def subset-trans sup-ge1*)
have *4*: (*X* ∩ *SInf*)·(*Y* ∩ *SFinite*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
using *Powerstarhelp2 sfinite-def* **by** *fastforce*
have *5*: (*X* ∩ *SInf*)·(*Y* ∩ *SInf*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
by (*metis 4 Powerstarhelp2*)
have *6*: (*X*·*Y*) ∩ *SFinite* = ((*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*) ∩ *SFinite*) ∪
((*X* ∩ *SFinite*)·(*Y* ∩ *SInf*) ∩ *SFinite*) ∪
((*X* ∩ *SInf*)·(*Y* ∩ *SFinite*) ∩ *SFinite*) ∪
((*X* ∩ *SInf*)·(*Y* ∩ *SInf*) ∩ *SFinite*)
using *1* **by** *auto*
from *6 2 3 4 5* **show** *?thesis* **by** *auto*

qed

lemma *FI04*:
$(X \cap SFinite)\cdot(Y \cap SFinite) = (X\cdot Y) \cap SFinite$
using *FI03 FI02* **by** *fastforce*

lemma *FI05*:
$SAlways\ SMore \subseteq SInf$
by (*metis B04 BD56 Compl-disjoint2 sfalse-def sinf-def smore-def sup.orderE*)

lemma *FI06*:
$SEmpty \subseteq SFinite$
using *BD01 BD02* **by** *auto*

lemma *FI07*:
$SSkip\cdot SFinite \subseteq SFinite$
using *SStarSkip UnfoldL* **by** *fastforce*

lemma *FI08*:
$SFinite\cdot SSkip \subseteq SFinite$
by (*metis FI07 SStar19 SStarSkip*)

lemma *FI09*:
$SFinite\cdot SSkip = SFinite \cap SMore$
proof −
have *1*: $SFinite\cdot SSkip \subseteq SFinite \cap SMore$
by (*metis B16 FI07 N09 N10 SStar19 SStarSkip*)
have *2*: $SFinite \cap SMore \subseteq SFinite\cdot SSkip$
by (*metis B20 FI01 FI02 SStar19 SStarSkip UnfoldL inf.idem smore-def*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *FI10*:
$SFinite\cdot\ SSkip = SSkip\ \cdot SFinite$
by (*metis SStar19 SStarSkip*)

lemma *FI11*:
$SFinite \cap SEmpty = SEmpty$
by (*simp add*: *FI06 Int-absorb2 inf-sup-aci*(*1*))

lemma *FI12*:
$SInf\cdot SInf = SInf$
by (*metis FusionSFalse Powerstarhelp2 sinf-def*)

lemma *FI13*:
$SFinite\cdot SInf = SInf$
by (*metis BD06 FusionAssoc1 sinf-def ssometime-def*)

lemma *FI14*:
$SInf\ \cdot\ SFinite = SInf$

by (*metis FusionSFalse Powerstarhelp2 sinf-def*)

lemma *FI15*:
SInf = − *SFinite*
by (*simp add*: *sfinite-def*)

lemma *FI16*:
SFinite = − *SInf*
by (*simp add*: *sfinite-def*)

lemma *FI17*:
((*F*·*STrue*) ∩ *SFinite*) = (*F* ∩ *SFinite*)·*SFinite*
by (*metis FI04 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

lemma *FI18*:
((*STrue*·*F*) ∩ *SFinite*) = *SFinite*·(*F* ∩ *SFinite*)
by (*metis BD19 FI01 FI04 FI17 FusionSEmptyR inf.idem sda-def*)

lemma *FI19*:
SFinite· *SMore* = *SMore*
by (*metis BD06 FusionAssoc1 N14 N15 ssometime-def subset-antisym*)

lemma *FI20*:
SInf ∪ *SFinite* = *STrue*
using *STrueTop sfinite-def* **by** *auto*

lemma *FI21*:
SFMore = *SSkip*·*SFinite*
using *FI09 FI10 sfmore-def* **by** *auto*

lemma *FI22*:
(*F* ∩ *SFinite* ⊆ *G*) = ((*F* ∩ *SFinite*) ⊆ (*G* ∩ *SFinite*))
by *simp*

lemma *FI23*:
(*F*·*G*) ∩ *SInf* = *F* · (*G* ∩ *SInf*)
by (*metis FusionAssoc FusionSFalse*)

lemma *FI24*:
((*F*·*G*) ∩ *SInf*) = ((*F* ∩ *SInf*) ∪ ((*F* ∩ *SFinite*)·(*G* ∩ *SInf*)))
using *FI23*[*of F G*] *Powerstarhelp2 Powerstarhelp3* **by** *fastforce*

lemma *FI25*:
SMore ∩ *SInf* = *SInf*
using *SStar15 SStarSkip sfinite-def smore-def* **by** *fastforce*

lemma *FI26*:
(*F* ∩ *SInf*)·(*F* ∩ *SInf*) = (*F* ∩ *SInf*)
using *Powerstarhelp2* **by** *blast*

lemma *FI27*:
$(F \cap SInf)\cdot G = (F \cap SInf)$
using *Powerstarhelp2* **by** *blast*

lemma *FI28*:
$((F \cap SMore) \cap SInf) = (F \cap SInf)$
using *FI25* **by** *blast*

lemma *FI29*:
$((F \cap SMore) \cap SFinite) = (F \cap SFMore)$
by (*metis inf-assoc inf-commute sfmore-def*)

lemma *FI30*:
$(SEmpty \cap SFMore) = SFalse$
by (*simp add*: *sfalse-def sfmore-def smore-def*)

lemma *FI31*:
$((F \cap SInf) \cap SFMore) = SFalse$
by (*simp add*: *sfalse-def sfinite-def sfmore-def*)

lemma *FI32*:
$(SEmpty \cap SInf) = SFalse$
using *FI25 sfalse-def smore-def* **by** *auto*

lemma *FI33*:
$(F \cap SInf) = F\cdot SFalse$
by (*simp add*: *FusionSFalse*)

lemma *FI34*:
$(F \cap SFinite)\cdot G \subseteq SSometime\ G$
by (*simp add*: *FusionRuleL ssometime-def*)

lemma *FI35*:
assumes $F \subseteq SNext\ F$
shows $SFinite \subseteq -F$
using *assms*
by (*metis B01 FusionSEmptyR N12 N22 SStarInductL SStarSkip double-compl snext-def*)

lemma *FI36*:
assumes $F \cap -G \subseteq SNext\ F$
shows $F \cap SFinite \subseteq SSometime\ G$
using *assms*
proof −
have *1*: $F \cap -G \subseteq SNext\ F$
using *assms* **by** *auto*
have *2*: $F \cap -G \cap SAlways\ (-G) \subseteq SNext\ F \cap SAlways\ (-G)$
using *assms* **by** *blast*
have *3*: $SAlways\ (-G) \subseteq -G$
by (*simp add*: *BD12*)
have *4*: $SAlways\ (-G) = SAlways\ (-G) \cap -G$

using *3* **by** *blast*
have *5*: $F \cap SAlways\ (-G) \subseteq SNext\ F \cap SAlways\ (-G)$
using *2 4* **by** *blast*
have *51*: $(SFinite \cdot G) = G \cup (SSkip \cdot (SFinite \cdot G))$
by (*metis FusionAssoc1 SStarSkip ST47 UnfoldL*)
have *6*: $SAlways\ (-G) = -G \cap SWnext\ (SAlways\ (-G))$
using *51* **by** (*simp add*: *salways-def ssometime-def swnext-def*) *blast*
have *7*: $SNext\ F \cap SAlways\ (-G) \subseteq SNext\ F \cap SWnext\ (SAlways\ (-G))$
using *6* **by** *blast*
have *8*: $F \cap SAlways\ (-G) \subseteq SNext\ F \cap SWnext\ (SAlways\ (-G))$
using *5 7* **by** *blast*
have *9*: $F \cap SAlways\ (-G) \subseteq SMore \cap SWnext\ F \cap SWnext\ (SAlways\ (-G))$
using *8 N16* **by** *blast*
have *10*: $F \cap SAlways\ (-G) \subseteq SWnext\ F \cap SWnext\ (SAlways\ (-G))$
using *8 N16* **by** *fastforce*
have *11*: $F \cap SAlways\ (-G) \subseteq SWnext\ (F \cap SAlways\ (-G))$
using *10 N17* **by** *blast*
have *12*: $SAlways\ (F \cap SAlways\ (-G)) \subseteq SWnext\ (F \cap SAlways\ (-G))$
by (*metis 10 B15 BD12 N17 inf.absorb-iff2*)
have *13*: $SAlways\ (F \cap SAlways\ (-G)) \subseteq$
$SWnext\ (F \cap SAlways\ (-G)) \cap F \cap (-(SAlways\ (-G)) \cup SAlways(\ F \cap SAlways\ (-G)))$
using *12 BD12* **by** *auto*
have *14*: $(F \cap SAlways\ (-G)) \subseteq SAlways(\ F \cap SAlways\ (-G))$
using *12 13*
by (*metis 10 B08 BD08 N17 SBoxInduct boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)
have *15*: $SAlways(\ F \cap SAlways\ (-G)) \subseteq (\ F \cap SAlways\ (-G))$
using *BD12* **by** *blast*
have *16*: $SAlways\ (F \cap SAlways\ (-G)) = (\ F \cap SAlways\ (-G))$
using *14 15* **by** *blast*
have *17*: $(\ F \cap SAlways\ (-G)) \subseteq SMore$
using *9* **by** *blast*
have *18*: $SAlways\ (F \cap SAlways\ (-G)) \subseteq SAlways\ SMore$
by (*simp add*: *17 BD38*)
have *19*: $SFinite = -(SAlways\ SMore)$
by (*simp add*: *BD01 salways-def smore-def*)
have *20*: $SFinite \subseteq -(F \cap\ SAlways\ (-G))$
using *16 18 19* **by** *blast*
have *21*: $SFinite \subseteq -F \cup -(SAlways\ (-G))$
using *20* **by** *blast*
have *22*: $-(SAlways\ (-G)) = SSometime\ G$
by (*simp add*: *BD09*)
show *?thesis*
using *20 22* **by** *blast*
qed

lemma *FI37*:
assumes $F \cap -G \subseteq SNext\ (F \cap -G)$
shows $F \cap SFinite \subseteq SSometime\ G$
using *assms*
by (*metis B15 FI36 N05*)

lemma *FI38*:
assumes $F \cap -G \subseteq (SNext\ F) \cap -(SNext\ G)$
shows $F \cap SFinite \subseteq G$
proof −
have *1*: $F \cap -G \subseteq SNext((F \cap -G))$
using *N17*[*of* $F\ -G$]
by (*metis* (*no-types*, *lifting*) *N12 N16 assms double-compl inf.semigroup-axioms semigroup.assoc*)
have *2*: $SFinite \subseteq -((F \cap -G))$
using *1 FI35* **by** *auto*
show *?thesis*
using *2* **by** *blast*
qed

lemma *FI39*:
assumes $SWnext\ (SSometime\ F) \subseteq F$
shows $SFinite \subseteq F$
proof −
have *1*: $-F \subseteq SNext\ (-F)$
by (*metis BD02 N12 N18 Subsumption assms compl-le-swap2 double-complement sup.coboundedI1*)
from *1* **show** *?thesis* **using** *FI35* **by** *blast*
qed

lemma *FI40*:
assumes $SEmpty \subseteq F$
$SNext\ F \subseteq F$
shows $SFinite \subseteq F$
proof −
have *1*: $-F \subseteq SNext\ (-F)$
using *N12 N19 assms*(*1*) *assms*(*2*) **by** *blast*
from *1* **show** *?thesis* **using** *FI35* **by** *blast*
qed

lemma *FI41*:
assumes $SEmpty \cap F \subseteq G$
$SNext\ (-F \cup G) \cap F \subseteq G$
shows $F \cap SFinite \subseteq G$
proof −
have *1*: $(F \cap -G) \subseteq SNext\ (F \cap -G)$
using *N19*[*of* $(-F \cup G)$]
using *assms*(*1*) *assms*(*2*) *snext-def swnext-def* **by** *fastforce*
have *2*: $SFinite \subseteq -\ (F \cap -G)$
using *1 FI35* **by** *auto*
from *2* **show** *?thesis* **by** *blast*
qed

lemma *FI42*:
assumes $F \subseteq SFMore\cdot\ F$
shows $SFinite \subseteq -F$
proof −

have *1*: *SSometime F* ⊆ *SNext* (*SSometime F*)
by (*simp add*: *ssometime-def snext-def*)
(*metis FI21 SChopAssoc SStarSkip ST47 UnfoldL assms order-refl subset-Un-eq*)
have *2*: *SFinite* ⊆ −(*SSometime F*)
by (*simp add*: *1 FI35*)
show *?thesis*
by (*metis 2 B01 FI21 SStarSkip ST47 UnfoldL assms le-iff-sup ssometime-def subset-trans*)
qed

lemma *FI43*:
assumes *F* ∩ −*G* ⊆ *SFMore*·(*F* ∩ −*G*)
shows *F* ∩ *SFinite* ⊆ *G*
proof −
have *1*: *SFinite* ⊆ −(*F* ∩ −*G*)
using *FI42 assms* **by** *blast*
from *1* **show** *?thesis* **by** *blast*
qed

lemma *FI44*:
assumes *F* ∩ *SFinite* ⊆ *SFMore* · *F*
shows *SFinite* ⊆ −*F*
proof −
have *1*: *F* ∩ *SFinite* ⊆ *SFMore* · (*F* ∩ *SFinite*)
by (*metis FI04 FI21 FI22 SSkipSFinite assms inf.idem*)
show *?thesis*
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma *FI45*:
assumes *F* ∩ *SFinite* ⊆ *SMore* · *F*
shows *SFinite* ⊆ −*F*
proof −
have *1*: *F* ∩ *SFinite* ⊆ *SFMore* · (*F* ∩ *SFinite*)
by (*metis FI04 FI22 assms inf-commute sfmore-def*)
show *?thesis*
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma *FI46*:
assumes *F* ⊆ *SMore* · *F*
shows *SFinite* ⊆ −*F*
proof −
have *1*: *F* ∩ *SFinite* ⊆ *SFMore* · (*F* ∩ *SFinite*)
using *B11 FI45 assms* **by** *auto*
show *?thesis*
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma *FI47*:
assumes (*F* ∩ −*G*) ∩ *SFinite* ⊆ *SFMore* · (*F* ∩ −*G*)

shows $F \cap SFinite \subseteq G$
proof −
have *1*: $SFinite \subseteq -(\ F \cap -G)$
using *FI44 assms* **by** *blast*
from *1* **show** *?thesis* **by** *blast*
qed

lemma *FI48*:
assumes $(F \cap -G)\ \subseteq SMore \cdot (F \cap -G)$
shows $F \cap SFinite \subseteq G$
proof −
have *1*: $SFinite \subseteq -(\ F \cap -G)$
using *FI45 assms* **by** *blast*
from *1* **show** *?thesis* **by** *blast*
qed

lemma *FI49*:
assumes $F \subseteq G \cdot F$
$G \subseteq SFMore$
shows $SFinite \subseteq -F$
proof −
have *1*: $G{\cdot}F \subseteq SFMore \cdot F$
by (*simp add*: *FusionRuleL assms*(*2*))
have *2*: $F \subseteq SFMore \cdot F$
using *1 assms*(*1*) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *FI42*)
qed

lemma *FI50*:
assumes $F \subseteq G \cdot F$
$G \subseteq SMore$
shows $SFinite \subseteq -F$
proof −
have *1*: $G{\cdot}F \subseteq SMore \cdot F$
by (*simp add*: *FusionRuleL assms*(*2*))
have *2*: $F \subseteq SMore \cdot F$
using *1 assms*(*1*) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *FI46*)
qed

lemma *FI51*:
assumes $F \cap -G \subseteq (H{\cdot}F) \cap -(H{\cdot}G)$
$H \subseteq SFMore$
shows $F \cap SFinite \subseteq G$
proof −
have *1*: $H{\cdot}(F \cap -G) \subseteq SFMore \cdot (F \cap -G)$
by (*simp add*: *FusionRuleL assms*(*2*))
have *2*: $F \cap -G \subseteq SFMore \cdot (F \cap -G)$

using *1 CH02 assms(1)* **by** *blast*
from *2* **show** *?thesis*
by (*simp add*: *FI43*)
qed

lemma *FI52*:
assumes $F \cap -G \subseteq (H \cdot F) \cap -(H \cdot G)$
$H \subseteq SMore$
shows $F \cap SFinite \subseteq G$
proof −
have *1*: $H \cdot (F \cap -G) \subseteq SMore \cdot (F \cap -G)$
by (*simp add*: *FusionRuleL assms(2)*)
have *2*: $F \cap -G \subseteq SMore \cdot (F \cap -G)$
using *1 CH02 assms(1)* **by** *blast*
from *2* **show** *?thesis*
by (*simp add*: *FI48*)
qed

lemma *FI53*:
$SAlways\ SInf = SInf$
by (*metis BD01 BD35 BD42 FI15 salways-def*)

Omega

lemma *OA01*:
$(somega\ SSkip) = SSkip \cdot (somega\ SSkip)$
by (*metis SOmegaUnroll SSkipSFinite SSkipSMore*)

lemma *OA02*:
$(somega\ SEmpty) = SFalse$
by (*metis FI30 SFalseFusion SOmegaUnroll inf-assoc inf-commute sfmore-def*)

lemma *OA03*:
$(somega\ SFalse) = SFalse$
by (*metis Compl-disjoint2 Powerstarhelp2 SOmegaUnroll inf-assoc inf-compl-bot-right sfalse-def*)

lemma *OA04*:
$(somega\ SInf) = SFalse$
by (*metis FI25 SFalseBottom SFalseFusion SOmegaUnroll inf-commute sfinite-def*)

lemma *BD59*:
$SInf \cap G \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq$
$((F \cap SMore) \cap SFinite) \cdot (SInf \cap G \cap SAlways(-G \cup (((F \cap SMore) \cap SFinite) \cdot G)))$
proof −
have *1*: $SInf \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq (-G \cup (((F \cap SMore) \cap SFinite) \cdot G))$
using *BD12* **by** *blast*
have *2*: $SInf \cap G \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq$
$SInf \cap (((F \cap SMore) \cap SFinite) \cdot G) \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite) \cdot G))$
using *1* **by** *blast*
have *3*: $SInf \cap (((F \cap SMore) \cap SFinite) \cdot G) \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq$

$((F \cap SMore)\cap SFinite)\cdot(\ SInf \cap G \cap SAlways(-G \cup (((F \cap SMore)\cap SFinite)\cdot G)))$
by (*metis BD58 FI23 inf-commute*)
show *?thesis* **using** *2 3* **by** *blast*
qed

lemma *OA06*:
$SInf \cap G \cap SAlways\ (\ -G \cup (((F \cap SMore)\cap SFinite)\cdot G)) \subseteq (somega\ F)$
by (*metis* (*no-types*, *lifting*) *BD59 SOmegaWeakCoinduct*)

lemma *FI54*:
$SAlways\ SFMore \subseteq SInf$
by (*metis BD38 BD56 Compl-disjoint2 SMoreImpSSkipFusion SSkipFusionImpSMore inf-le2 sfalse-def sfmore-def sinf-def smore-def subset-antisym sup.orderE*)

lemma *FI55*:
$SAlways\ SFinite = SFinite$
using *FI13 FI15 salways-def ssometime-def* **by** *fastforce*

lemma *FI56*:
$(SAlways\ SFMore) = SFalse$
using *BD12 FI31 FI54* **by** *blast*

lemma *OA07*:
$(somega\ ((SPower\ SSkip\ (Suc\ n)))) \subseteq SInf$
by (*metis FI15 FI42 FusionRuleL SOmegaUnroll boolean-algebra-cancel.inf0 compl-le-swap1 inf-commute inf-le2 inf-mono sfmore-def*)

lemma *OA08*:
$SInf \subseteq (somega\ ((SPower\ SSkip\ (Suc\ n))))$
proof −
have *1*: $SInf \cap STrue \cap SAlways\ (\ -STrue \cup ((((SPower\ SSkip\ (Suc\ n)) \cap SMore)\cap SFinite)\cdot STrue)) \subseteq (somega\ (SPower\ SSkip\ (Suc\ n)))$
by (*simp add*: *OA06*)
have *2*: $-\ STrue \cup (SPower\ SSkip\ (Suc\ n) \cap SMore)\cap SFinite \cdot STrue = (SPower\ SSkip\ (Suc\ n) \cap SMore)\cap SFinite \cdot STrue$
by (*simp add*: *strue-def*)
have *21*: $((((SPower\ SSkip\ (Suc\ n)) \cap SMore)\cap SFinite)\cdot STrue) \subseteq SMore$
by (*metis N13 N14 N15 SStar25 SStarInductR inf-le2 le-sup-iff subset-antisym sup-inf-absorb*)
have *3*: $SAlways\ (\ ((((SPower\ SSkip\ (Suc\ n)) \cap SMore)\cap SFinite)\cdot STrue)) \subseteq SAlways\ (SMore)$
using *BD38 21* **by** *auto*
have *4*: $SAlways\ (SMore) \subseteq SInf$
using *FI05* **by** *blast*
have *5*: $SAlways\ (\ -STrue \cup ((((SPower\ SSkip\ (Suc\ n)) \cap SMore)\cap SFinite)\cdot STrue)) \subseteq SInf$
by (*metis 2 3 FI05 subset-trans*)
have *6*: $SInf \subseteq SAlways\ (SMore)$
by (*simp add*: *BD01 FI15 salways-def smore-def*)
have *7*: $(SPower\ SSkip\ (Suc\ n)) \cap SMore \cap SFinite \subseteq SMore \cap SFinite$
using *FI07 FI21* **by** *blast*
have *8*: $SMore \subseteq (SMore \cap SFinite)\cdot STrue$
by (*metis FI19 FI21 ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusionImpSMore SStar19 SStarSkip*

inf-commute sfmore-def subset-antisym)
have *9*: *SInf* ∩ ((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*STrue*) =
(((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*SInf*)
by (*metis FI23 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)
have *10*: *SInf* ∩ *SAlways* (((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*STrue*)) =
SAlways (((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*SInf*))
by (*metis 9 BD29 FI53*)
have *11*: ((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*SInf*) = *SInf*
proof (*induct n*)
case *0*
then show *?case*
proof −
have *f1*: (*SFinite*) ∩ *SSkip* = *SSkip*
using *SSkipSFinite* **by** *blast*
have *f2*: ∀ *S*. *S* · *SSkip* ⊆ (*SMore* ∩ *SFinite*) ∨ ¬ *S* ⊆ *SFinite*
using *FI10 FI21 FusionRuleL* **by** (*metis inf-commute sfmore-def*)
have *f3*: (*SEmpty*) · *SSkip* = *SSkip*
using *f1* **by** (*metis FusionSEmptyR inf-commute pwr-0 spower-commutes*)
have *f4*: (*SEmpty*) ⊆ *SFinite*
using *pwr-0 spower-finite* **by** *blast*
have *f5*: (*SSkip*) ⊆ *SFinite*
using *f1* **by** *blast*
have *f6*: (*SSkip*) · (*STrue* · (*SFalse* · *SFalse*)) = *SInf*
by (*metis FI25 FusionAssoc1 FusionSFalse Powerstarhelp2 SMoreImpSSkipFusion SSkipFusion-ImpSMore*
subset-antisym)
have *f7*: (*SSkip*) ∩ *SMore* ∩ *SFinite* = *SSkip*
using *f4 f3 f2* **by** *blast*
have (*SSkip*) · *SInf* = *SInf*
using *f6 f5 f4 f3 f2* **by** (*simp add*: *SFalseFusion sinf-def*)
then show *?thesis*
by (*simp add*: *f7 FusionSEmptyR SSkipSFinite*)
qed
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: *SSkip* · *STrue* = *SMore*
using *SMoreImpSSkipFusion SSkipFusionImpSMore* **by** *blast*
have *f2*: ∀ *S*. *STrue* ∩ *S* = *S*
by (*simp add*: *strue-def*)
have *f3*: ∀ *S*. *SMore* ∩ (*SSkip* · *S*) = *SSkip* · *S*
using *f1*
by (*metis* (*no-types*) *B14 CH06 boolean-algebra-cancel.inf0 compl-bot-eq inf-aci*(*1*) *inf-le2 strue-def*)
then have *f0*: ∀ *S*. *SMore* ∩ *SFinite* ∩ (*SSkip* · *S*) = *SFinite* ∩ (*SSkip* · *S*)
by *blast*
then show *?thesis*
using *f3 f2 f1*
proof −
have *f4*: *SPower* (*SSkip*) (*Suc* (*Suc n*)) ∩ *SMore* ∩ *SFinite*= *SFinite* ∩ (*SSkip* ∩ *SFinite* · *SPower*

SSkip (*Suc n*))
by (*metis SSkipSFinite f3 inf-sup-aci*(*1*) *pwr-Suc*)
then have *f5*: *SPower* (*SSkip*) (*Suc* (*Suc n*)) ∩ *SMore* ∩ *SFinite* = *SSkip* ∩ *SFinite* · *SPower SSkip* (*Suc n*)
proof −
have (*SSkip*) ∩ *SFinite* · *SPower SSkip* (*Suc n*) ⊆ *SFinite*
using *pwr-Suc spower-finite* **by** *blast*
then show *?thesis*
using *f4* **by** *blast*
qed
show *?thesis*
by (*metis B14 CH01 FI12 FI23 FI25 FusionAssoc1 Powerstarhelp3 SSkipSFinite Suc f1 f3*
inf-commute pwr-Suc sinf-def spower-finite sup-inf-absorb)
qed
qed
qed
have *12*: *SInf* ⊆ *SAlways* (((((*SPower SSkip* (*Suc n*)) ∩ *SMore*) ∩*SFinite*)·*SInf*))
by (*metis 11 B04 FI53*)
show *?thesis*
using *11 SOmegaWeakCoinduct* **by** (*metis 12 FI53*)
qed

lemma *OA09*:
(*somega* ((*SPower SSkip* (*Suc n*)))) = *SInf*
using *OA07 OA08* **by** *blast*

lemma *OA10*:
(*somega SSkip*) = *SInf*
by (*metis FusionSEmptyR OA09 SSkipSFinite pwr-0 spower.simps*(*2*))

lemma *OA11*:
(*somega STrue*) ⊆ *SInf*
by (*metis FI15 FI42 SOmegaUnroll boolean-algebra.compl-zero boolean-algebra-cancel.inf0*
compl-le-swap1 dual-order.refl inf-commute sfmore-def strue-def)

lemma *OA12*:
SInf ⊆ (*somega STrue*)
proof −
have *1*: (*SAlways* (*SFalse* ∪ ((*STrue* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ *SInf*
by (*metis B04 BD01 BD09 BD35 FI10 FI15 FI19 FI21 ITA.SChopAssoc SMoreImpSSkipFusion*
SSkipFusionImpSMore boolean-algebra.compl-zero boolean-algebra.de-Morgan-disj
boolean-algebra-cancel.inf0 inf-commute salways-def sfalse-def sfmore-def smore-def strue-def)
have *2*: *SInf* ⊆ (*SAlways* (*SFalse* ∪ ((*STrue* ∩ *SMore*)∩*SFinite*)·*STrue*))
by (*metis 1 B15 BD01 BD09 BD12 BD35 FI10 FI15 FI19 FI21 FI56 ITA.SChopAssoc SMoreImpSSkip-Fusion*
SSkipFusionImpSMore boolean-algebra.compl-zero boolean-algebra-cancel.inf0 inf.absorb-iff2
inf-commute salways-def sfmore-def smore-def strue-def sup.absorb2)
have *3*: *SInf* ∩ *STrue* ∩ (*SAlways* (*SFalse* ∪ ((*STrue* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ (*somega STrue*)

by (*metis OA06 compl-bot-eq compl-top-eq sfalse-def strue-def*)
show *?thesis*
using *2 3 strue-def* **by** *fastforce*
qed

lemma *OA13*:
(*somega STrue*) = *SInf*
by (*simp add*: *B04 OA11 OA12*)

lemma *OA14*:
(*somega SMore*) ⊆ *SInf*
by (*metis Int-absorb1 Int-lower2 OA13 SOmegaUnroll SOmegaWeakCoinduct compl-bot-eq inf-top-left strue-def*)

lemma *OA15*:
SInf ⊆ (*somega SMore*)
proof −
have *1*: (*SAlways* (*SFalse* ∪ ((*SMore* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ *SInf*
by (*metis FI05 FI10 FI19 FI21 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusion-ImpSMore*
ST33 boolean-algebra.compl-one boolean-algebra.compl-zero boolean-algebra.de-Morgan-conj
inf.idem inf-commute sfalse-def sfmore-def smore-def strue-def subset-antisym)
have *2*: *SInf* ⊆ (*SAlways* (*SFalse* ∪ ((*SMore* ∩ *SMore*)∩*SFinite*)·*STrue*))
by (*metis BD38 FI10 FI19 FI21 FI25 FI53 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion*
SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero
boolean-algebra.de-Morgan-conj inf.idem inf-commute inf-le1 sfalse-def sfmore-def
smore-def strue-def subset-antisym)
have *3*: *SInf* ∩ *STrue* ∩ (*SAlways* (*SFalse* ∪ ((*SMore* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ (*somega STrue*)
using *OA13* **by** *blast*
show *?thesis*
by (*metis 1 2 Int-absorb1 OA06 boolean-algebra-cancel.inf0 compl-bot-eq compl-top-eq sfalse-def*
strue-def subset-antisym)
qed

lemma *OA16*:
(*somega SMore*) = *SInf*
using *OA14 OA15* **by** *blast*

lemma *OA17*:
(*somega SFinite*) ⊆ *SInf*
by (*metis FI07 FI15 FI21 FI42 SOmegaUnroll compl-le-swap1 dual-order.refl inf.orderE sfmore-def*)

lemma *OA18*:
SInf ⊆ (*somega SFinite*)
proof −
have *1*: (*SAlways* (*SFalse* ∪ ((*SFinite* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ *SInf*
by (*metis B14 FI05 FI07 FI10 FI19 FI21 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion*
SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero
boolean-algebra.de-Morgan-conj sfalse-def sfmore-def smore-def strue-def subset-antisym)
have *2*: *SInf* ⊆ (*SAlways* (*SFalse* ∪ ((*SFinite* ∩ *SMore*)∩*SFinite*)·*STrue*))
by (*metis B14 BD38 FI10 FI19 FI21 FI25 FI53 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion*

SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero boolean-algebra.de-Morgan-conj inf-le1 sfalse-def sfmore-def smore-def strue-def subset-antisym)
have *3*: $SInf \cap STrue \cap (SAlways\ (SFalse \cup ((SFinite \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (somega\ STrue)$
using *OA13* **by** *blast*
show *?thesis*
by (*metis 1 2 Int-absorb1 OA06 boolean-algebra-cancel.inf0 compl-bot-eq compl-top-eq sfalse-def strue-def subset-antisym*)
qed

lemma *OA19*:
$(somega\ SFinite) = SInf$
by (*simp add*: *B04 OA17 OA18*)

lemma *OA20*:
assumes $H \subseteq ((F \cap SMore) \cap SFinite) \cdot H$
shows $H \cap SInf \subseteq (somega\ F)$
using *assms*
using *SOmegaWeakCoinduct* **by** *blast*

lemma *OA21*:
assumes $F \subseteq G$
shows $(somega\ F) \cap SInf \subseteq (somega\ G)$
using *assms*
by (*metis FusionRuleL OA20 SOmegaUnroll inf-mono subset-refl*)

lemma *OA22*:
assumes $F = G$
shows $(somega\ F) = (somega\ G)$
using *assms* **by** *auto*

lemma *OA23*:
$(somega\ (F \cap G)) \cap SInf \subseteq (somega\ F)$
by (*simp add*: *OA21*)

lemma *OA24*:
$(somega\ (F \cap G)) \cap SInf \subseteq (somega\ G)$
by (*simp add*: *OA21*)

lemma *BD60*:
$(SBi\ F) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
proof −
have *1*: $F \subseteq (-G0 \cup (F \cap G0))$
by *blast*
have *2*: $SBi\ F \subseteq SBi(-G0 \cup (F \cap G0))$
by (*simp add*: *1 BD39*)
have *3*: $SBi(-G0 \cup (F \cap G0)) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
by (*simp add*: *BD45*)
show *?thesis*
using *2 3* **by** *blast*
qed

lemma *BD61*:
$(SAlways\ H) \cap (F{\cdot}G) \subseteq F{\cdot}(H \cap G)$
proof −
have *1*: $H \subseteq (-G \cup (H \cap G))$
by *blast*
have *2*: $SAlways\ H \subseteq SAlways\ (-G \cup (H \cap G))$
by (*simp add*: *1 BD38*)
have *3*: $SAlways\ (-G \cup (H \cap G)) \cap (F{\cdot}G) \subseteq F{\cdot}(H \cap G)$
using *BD46* **by** *blast*
show *?thesis*
using *2 3* **by** *blast*
qed

lemma *BD62*:
$(SBa\ F) \cap (G0{\cdot}G1) \subseteq (F \cap G0){\cdot}(F \cap G1)$
proof −
have *1*: $(SBa\ F) \subseteq (SBi\ F)$
by (*simp add*: *BD27*)
have *2*: $(SBi\ F) \cap (G0{\cdot}G1) \subseteq (F \cap G0){\cdot}G1$
by (*simp add*: *BD60*)
have *3*: $(SBa\ F) \subseteq (SAlways\ F)$
by (*simp add*: *BD28*)
have *4*: $(SAlways\ F) \cap ((F \cap G0){\cdot}G1) \subseteq (F \cap G0){\cdot}(F \cap G1)$
using *BD61* **by** *blast*
show *?thesis*
using *1 2 3 4* **by** *blast*
qed

lemma *BD63*:
$(SBa\ F) \cap (G \cdot G1) \subseteq (F \cap G)\ {\cdot}((SBa\ F)\ \cap G1)$
proof −
have *1*: $(SBa\ F) = (SBa\ (SBa\ F))$
using *BD44* **by** *blast*
have *2*: $(SBa\ (SBa\ F))\ \cap (G \cdot G1) \subseteq G{\cdot}((SBa\ F) \cap G1)$
using *BD28 BD61* **by** *blast*
have *3*: $(SBa\ F) \cap (G{\cdot}((SBa\ F) \cap G1)) \subseteq (F \cap G)\ {\cdot}((SBa\ F)\ \cap G1)$
using *BD62 FusionRuleR* **by** *blast*
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *OA25*:
$SBa\ (-F \cup G) \cap SInf \cap (somega\ F) \subseteq (somega\ G)$
proof −
have *1*: $SBa\ (-F \cup G) \cap (((F \cap SMore){\cap}SFinite){\cdot}(somega\ F)) \subseteq$
$((-F \cup G) \cap ((F \cap SMore){\cap}SFinite)){\cdot}((-F \cup G) \cap (somega\ F))$
by (*simp add*: *BD62*)
have *2*: $(-F \cup G) \cap ((F \cap SMore){\cap}SFinite) \subseteq ((G \cap SMore){\cap}SFinite)$
by *auto*

have *3*: $(-F \cup G) \cap (somega\ F) \subseteq (somega\ F)$
by *auto*
have *4*: $SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq ((G \cap SMore) \cap SFinite) \cdot (somega\ F)$
using *1 2 3 CH06* **by** *blast*
have *5*: $SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq$
$((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$
using *BD63* **by** *blast*
have *6*: $((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot (SBa\ (-F \cup G) \cap (somega\ F)) \subseteq$
$((G \cap SMore) \cap SFinite) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$
using *2 FusionRuleL* **by** *blast*
have *7*: $(SBa\ (-F \cup G) \cap (somega\ F)) \subseteq ((G \cap SMore) \cap SFinite) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$
using *5 6 SOmegaUnroll* **by** *blast*
have *8*: $(SBa\ (-F \cup G) \cap (somega\ F)) \cap SInf \subseteq (somega\ G)$
by (*simp add*: *7 OA20*)
show *?thesis*
using *8* **by** *auto*
qed

lemma *OA26*:
$SBa\ ((-F \cup G) \cap (-G \cup F)) \ \cap SInf \subseteq (-(somega\ G) \cup (somega\ F)) \cap (-(somega\ F) \cup (somega\ G))$
proof −
have *1*: $SBa\ ((-F \cup G) \cap (-G \cup F)) = SBa\ (-F \cup G) \cap SBa\ (-G \cup F)$
by (*simp add*: *BD22 sba-def*)
have *2*: $SBa\ (-F \cup G) \cap SInf \subseteq (-(somega\ F) \cup (somega\ G))$
by (*simp add*: *B19 OA25*)
have *3*: $SBa\ (-G \cup F) \cap SInf \subseteq (-(somega\ G) \cup (somega\ F))$
by (*simp add*: *B19 OA25*)
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *OA27*:
$SBa\ F \cap (somega\ G) \cap SInf \subseteq somega\ (F \cap G)$
by (*metis* (*no-types*, *lifting*) *BD63 OA20 SOmegaUnroll inf-assoc*)

lemma *FI57*:
$SInf \cap (((F \cap SMore) \cap SFinite) \cdot G) = ((F \cap SMore) \cap SFinite) \cdot (G \cap SInf)$
using *FI23* **by** *blast*

lemma *FI58*:
$SInf \cap (SAlways\ F) = SAlways\ (F \cap SInf)$
using *BD29 FI53* **by** *blast*

lemma *FI59*:
$SInf \cap (SAlways\ (-F \cup G)) = SAlways\ (\ (-F \cap SInf) \cup (G \cap SInf)\ \ )$
by (*simp add*: *FI58 inf-sup-distrib2*)

3.16.6 Link between Set of Intervals and ITL

lemma *interval-lan* [*simp*]:

$\sigma \in (lan\ f) \longleftrightarrow (\sigma \models f)$
by (*simp add*: *lan-def*)

lemma *valid-lan-eqv* :
$(\ (lan\ f) = (lan\ g)\ ) \longleftrightarrow (\ \vdash f = g\ )$
using *interval-lan lan-def Valid-def* **by** *fastforce*

lemma *valid-lan-imp* :
$(\ (lan\ f) \subseteq (lan\ g)\ ) \longleftrightarrow (\vdash f \longrightarrow g)$
using *interval-lan lan-def Valid-def*
by (*simp add*: *Valid-def lan-def Collect-mono-iff*)

lemma *valid-strue* :
$(\ (lan\ f) = STrue) \longleftrightarrow (\vdash f)$
using *strue-def* **by** *fastforce*

lemma *strue-true*:
$\sigma \in STrue \longleftrightarrow (\sigma \models \#True)$
by (*simp add*: *strue-elim*)

lemma *strue-true-1*:
$STrue = (lan\ (LIFT\ \#True))$
using *lan-def strue-true* **by** *fastforce*

lemma *sfalse-false*:
$\sigma \in SFalse \longleftrightarrow (\sigma \models \#False)$
by (*simp add*: *sfalse-def*)

lemma *sfalse-false-1*:
$SFalse = (lan\ (LIFT(\#False)))$
using *sfalse-false* **using** *lan-def* **by** *fastforce*

lemma *not-negation*:
$\sigma \in (-(lan\ f)) \longleftrightarrow (\sigma \models \neg\ f)$
by *simp*

lemma *not-negation-1*:
$-(lan\ f\ ) = (lan\ (LIFT(\neg\ f)))$
using *interval-lan lan-def* **by** *fastforce*

lemma *inter-and*:
$(\sigma \in (\ (lan\ f) \cap (lan\ g)\ )) \longleftrightarrow (\sigma \models f \wedge g)$
by (*simp add*: *lan-def*)

lemma *inter-and-1*:
$(\ (lan\ f) \cap (lan\ g)\ ) = (lan\ (LIFT(f \wedge g)))$
using *inter-and lan-def* **by** *fastforce*

lemma *union-or*:
$(\sigma \in (\ (lan\ f) \cup (lan\ g)\ )) \longleftrightarrow (\sigma \models f \vee g)$

by (*simp add*: *lan-def*)

lemma *union-or-1*:
((*lan f*) ∪ (*lan g*)) = (*lan* (*LIFT*(*f* ∨ *g*)))
using *union-or lan-def* **by** *fastforce*

lemma *subset-impl*:
(σ ∈ (−(*lan f*) ∪ (*lan g*))) ⟷ (σ ⊨ *f* ⟶ *g*)
by *simp*

lemma *subset-impl-1*:
(−(*lan f*) ∪ (*lan g*)) = (*lan* (*LIFT*(*f* ⟶ *g*)))
using *subset-impl lan-def* **by** *fastforce*

lemma *fusion-chop*:
(σ ∈ ((*lan f*)·(*lan g*))) ⟷ (σ ⊨ *f*;*g*)
by (*auto simp add*: *fusion-iff chop-nfuse*)

lemma *fusion-chop-1*:
((*lan f*)·(*lan g*)) = (*lan* (*LIFT*(*f*;*g*)))
using *fusion-chop lan-def* **by** *blast*

lemma *sempty-empty*:
σ ∈ *SEmpty* ⟷ (σ ⊨ *empty*)
by (*simp add*: *itl-defs sempty-elim zero-enat-def*)

lemma *sempty-empty-1*:
SEmpty = (*lan* (*LIFT empty*))
using *sempty-empty lan-def* **by** *fastforce*

lemma *smore-more*:
σ ∈ *SMore* ⟷ (σ ⊨ *more*)
using *zero-enat-def* **by** (*auto simp add*: *itl-defs sempty-elim smore-def*)

lemma *smore-more-1*:
SMore = (*lan* (*LIFT more*))
using *smore-more lan-def* **by** *fastforce*

lemma *sskip-skip*:
σ ∈ *SSkip* = (σ ⊨ *skip*)
by (*simp add*: *one-enat-def itl-defs sskip-elim*)

lemma *sskip-skip-1*:
SSkip = (*lan* (*LIFT skip*))
using *sskip-skip lan-def* **by** *fastforce*

lemma *snext-next*:
σ ∈ (*SNext* (*lan f*)) ⟷ (σ ⊨ ○ *f*)
by (*metis snext-def fusion-chop next-d-def sskip-skip-1*)

lemma *snext-next-1*:
$(SNext\ (lan\ f)) = (lan\ (LIFT(\bigcirc\ f)))$
using *snext-next lan-def* **by** *fastforce*

lemma *swnext-wnext*:
$\sigma \in (SWnext\ (lan\ f)) \longleftrightarrow (\sigma \models wnext\ f)$
by (*simp add*: *swnext-def fusion-chop-1 next-d-def not-negation-1 sskip-skip-1 wnext-d-def*)

lemma *swnext-wnext-1*:
$(SWnext\ (lan\ f)) = (lan\ (LIFT(wnext\ f)))$
using *swnext-wnext lan-def* **by** *fastforce*

lemma *sprev-prev*:
$\sigma \in (SPrev\ (lan\ f)) \longleftrightarrow (\sigma \models prev\ f)$
by (*metis fusion-chop prev-d-def sprev-def sskip-skip-1*)

lemma *sprev-prev-1*:
$(SPrev\ (lan\ f)) = (lan\ (LIFT(prev\ f)))$
using *sprev-prev lan-def* **by** *fastforce*

lemma *swprev-wprev*:
$\sigma \in (SWprev\ (lan\ f)) \longleftrightarrow (\sigma \models wprev\ f)$
by (*simp add*: *fusion-chop-1 not-negation-1 prev-d-def sskip-skip-1 swprev-def wprev-d-def*)

lemma *swprev-wprev-1*:
$(SWprev\ (lan\ f)) = (lan\ (LIFT(wprev\ f)))$
using *swprev-wprev lan-def* **by** *fastforce*

lemma *sinit-init*:
$\sigma \in SInit\ (lan\ f) \longleftrightarrow (\sigma \models init\ f)$
by (*simp add*: *Int-commute fusion-chop-1 init-d-def inter-and-1 sempty-empty-1 sinit-def strue-true-1*)

lemma *sinit-init-1*:
$SInit\ (lan\ f) = (lan\ (LIFT(init\ f)))$
using *sinit-init lan-def* **by** *fastforce*

lemma *sfinite*:
$\sigma \in SFinite \longleftrightarrow (\sigma \models finite)$
by (*simp add*: *sfinite-def sinf-def finite-d-def infinite-d-def fusion-chop sfalse-false-1 strue-true-1*)

lemma *sfinite-1*:
$SFinite = lan(LIFT(finite))$
using *sfinite lan-def* **by** *fastforce*

lemma *and-inter-finite*:
$\sigma \in (((lan\ f) \cap SFinite)) \longleftrightarrow (\sigma \models (f \wedge finite)\ )$
using *sfinite inter-and* **by** *auto*

lemma *and-inter-finite-1*:
$(((lan\ f) \cap SFinite)) = lan(LIFT\ (f \wedge finite)\ )$

by (*simp add*: *inter-and-1 sfinite-1*)

lemma *and-inter-more*:
$\sigma \in (((lan\ f) \cap SMore)) \longleftrightarrow (\sigma \models (f \wedge more)\ )$
using *smore-more inter-and* **by** *auto*

lemma *and-inter-more-1*:
$\sigma \in (((lan\ f) \cap SMore)) \longleftrightarrow (\sigma \in (lan\ (LIFT(f \wedge more))\ ))$
using *and-inter-more lan-def* **by** (*simp add*: *smore-more-1*)

lemma *and-inter-more-2*:
$((lan\ f) \cap SMore) = (lan\ (LIFT(f \wedge more))\ )$
using *and-inter-more-1* **by** *blast*

lemma *and-chop*:
$\sigma \in (((lan\ f) \cap SMore)\cdot (lan\ g)) \longleftrightarrow (\sigma \models (f \wedge more);g)$
by (*metis fusion-chop inter-and-1 smore-more-1*)

lemma *and-chop-1*:
$(((lan\ f) \cap SMore)\cdot (lan\ g)) = (lan\ (LIFT((f \wedge more);g)))$
using *and-chop lan-def* **by** *blast*

lemma *spower-chop-power*:
$(SPower\ (lan\ f)\ n) = (lan\ (LIFT(fpower\ f\ n)))$
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *sempty-empty-1 fpower-d-def*)
next
case (*Suc n*)
then show *?case*
by (*simp add*: *and-inter-finite-1 fusion-chop-1 fpower-d-def*)
qed

lemma *spowerstar*:
$\sigma \in SPowerstar\ (lan\ f) \longleftrightarrow \sigma \in SFPowerstar\ (lan\ (f)) \cdot (SEmpty \cup ((lan\ f) \cap SInf))$
by (*simp add*: *spowerstar-def*)

lemma *sstar-spowerstar*:
$\sigma \in SStar\ (lan\ f) \longleftrightarrow \sigma \in SPowerstar\ ((lan\ f) \cap SMore)$
by (*simp add*: *sstar-def*)

lemma *union-exists*:
$\sigma \in (\bigcup\ n.\ SPower\ (lan\ f)\ n) \longleftrightarrow \sigma \in lan(LIFT(\exists\, n.\ fpower\ f\ n))$
by (*simp add*: *spower-chop-power*)

lemma *union-exists-1*:
$(\bigcup\ n.\ SPower\ (lan\ f)\ n) = lan(LIFT(\exists\, n.\ fpower\ f\ n))$
using *union-exists lan-def* **by** *blast*

lemma *sstar-chopstar*:
$\sigma \in (SStar\ (lan\ f)) \longleftrightarrow \sigma \in (lan\ (LIFT(f^\star)))$
proof −
have *1*: $\sigma \in (SStar\ (lan\ f)) \longleftrightarrow \sigma \in SPowerstar\ ((lan\ f) \cap SMore)$
using *sstar-spowerstar* **by** *blast*
have *2*: $\sigma \in SPowerstar\ ((lan\ f) \cap SMore) \longleftrightarrow$
$\sigma \in SFPowerstar\ (lan\ (f) \cap SMore) \cdot (SEmpty \cup (((lan\ f) \cap SMore) \cap SInf))$
by (*simp add*: *spowerstar-def*)
have *3*: $SFPowerstar\ (lan\ (f) \cap SMore) = SFPowerstar(lan(LIFT(f \wedge more)))$
by (*simp add*: *and-inter-more-2*)
have *31*: $\bigwedge n.\ SPower\ (lan(LIFT(f \wedge more)))\ n = lan(LIFT(fpower\ (f \wedge more)\ n))$
using *spower-chop-power* **by** *blast*
have *32*: $SFPowerstar(lan(LIFT(f \wedge more))) = lan(LIFT(fpowerstar\ (f \wedge more)))$
using *union-exists-1* **by** (*auto simp add*: *sfpowerstar-def fpowerstar-d-def*)
have *4*: $(SEmpty \cup (((lan\ f) \cap SMore) \cap SInf)) =$
$lan(LIFT(empty \vee ((f \wedge more) \wedge inf)))$
by (*metis Morgan and-inter-more-2 finite-d-def inter-and-1 not-negation-1 sempty-empty-1*
sfinite-1 sfinite-def union-or-1)
have *5*: $SFPowerstar\ (lan\ (f) \cap SMore) \cdot (SEmpty \cup (((lan\ f) \cap SMore) \cap SInf)) =$
$lan(LIFT(powerstar\ (\ f \wedge more)))$
by (*simp add*: *powerstar-d-def 3 32 4 fpowerstar-d-def fusion-chop-1*)
have *6*: $lan(LIFT(powerstar\ (\ f \wedge more))) =$
$lan(LIFT(chopstar\ f))$
by (*simp add*: *chopstar-d-def*)
show *?thesis*
by (*simp add*: *1 2 5 6*)
qed

lemma *chopstar-sstar-1*:
$(SStar\ (lan\ f)) = (lan\ (LIFT(f^\star)))$
using *sstar-chopstar lan-def* **by** *blast*

lemma *chopstar-seqv*:
$\sigma \in (lan\ (LIFT(f^\star))) \longleftrightarrow \sigma \in (lan\ (LIFT(empty \vee\ (f \wedge more); f^\star)))$
by (*metis* (*no-types*, *lifting*) *SChopstarEqv chopstar-sstar-1 fusion-chop-1 inter-and-1*
sempty-empty-1 smore-more-1 union-or-1)

lemma *chopstar-seqv-1*:
$(lan\ (LIFT(f^\star))) = (lan\ (LIFT(empty \vee\ (f \wedge more); f^\star)))$
using *chopstar-seqv lan-def* **by** *blast*

lemma *sinf*:
$\sigma \in SInf \longleftrightarrow (\sigma \models inf)$
by (*simp add*: *fusion-chop infinite-d-def sfalse-false-1 sinf-def strue-true-1*)

lemma *sinf-1*:
$SInf = lan(LIFT(inf))$
using *sinf* **by** *fastforce*

lemma *fmore*:

$\sigma \in SFMore \longleftrightarrow (\sigma \models fmore)$
by (*metis fmore-d-def inf-commute inter-and-1 interval-lan sfinite-1 sfmore-def smore-more-1*)

lemma *fmore-1*:
$SFMore = lan(LIFT(fmore))$
using *fmore* **by** *fastforce*

lemma *omega-induct-sem*:
$x \in - (lan\ g) \cup (\ ((\ ((lan\ f) \cap SMore) \cap SFinite) \cdot (lan\ g))) \longleftrightarrow$
$(x \models g \longrightarrow ((f \wedge more) \wedge finite);g)$
by (*simp add*: *fusion-chop-1 inter-and-1 sfinite-1 smore-more-1*)

lemma *omega-induct*:
$- (lan\ g) \cup (\ ((\ ((lan\ f) \cap SMore) \cap SFinite) \cdot (lan\ g))) =$
$lan(LIFT(g \longrightarrow ((f \wedge more) \wedge finite);g))$
using *omega-induct-sem*[*of - g f*] **by** *fastforce*

lemma *somega-omega-sem-1*:
assumes $x \models f^\omega$
shows $x \in somega\ (lan\ f)$
proof −
have *1*: $x \models f^\omega \longrightarrow ((f \wedge more) \wedge finite);f^\omega$
by (*metis OmegaUnroll Valid-def int-iffD1 schop-d-def*)
have *2*: $x \in -(lan\ (LIFT(f^\omega))) \cup ((((lan\ f) \cap SMore) \cap SFinite) \cdot (lan(LIFT(f^\omega))))$
using *1 omega-induct-sem* **by** *blast*
have *3*: $\bigwedge x\ .\ x \in (lan\ (LIFT(f^\omega))) \Longrightarrow x \in (\ ((((lan\ f) \cap SMore) \cap SFinite) \cdot (lan\ (LIFT(f^\omega)))\ ))$
by (*metis* (*mono-tags*, *opaque-lifting*) *OmegaUnroll and-inter-finite-1 and-inter-more-2 fusion-chop-1 int-eq itl-def*(*9*))
show *?thesis* **using** *3 SOmegaWeakCoinductsem assms interval-lan* **by** *blast*
qed

lemma *somega-omega-sem-2*:
assumes $x \in somega\ (lan\ f)$
shows $x \models f^\omega$
proof −
have *1*: $x \in - (somega\ (lan\ f)) \cup (\ ((\ ((lan\ f) \cap SMore) \cap SFinite) \cdot (somega\ (lan\ f))))$
using *SOmegaCases* **by** *blast*
have *2*: $\bigwedge x.\ (\lambda\ x.\ x \in somega\ (lan\ f))\ x \Longrightarrow$
$x \in (\ ((\ ((lan\ f) \cap SMore) \cap SFinite) \cdot (somega\ (lan\ f))))$
using *SOmegaCases* **by** *blast*
have *3*: $\bigwedge x.\ (\lambda\ x.\ x \in somega\ (lan\ f))\ x \Longrightarrow$
$(\ (\exists\, n.\ f\ (\ (ntaken\ n\ \ x)) \wedge n > 0 \wedge$
$(\lambda\ x.\ x \in somega\ (lan\ f))\ (\ (ndropn\ n\ (\ x))))\ )$

using *interval-lan*[*of - f*] *somega.cases*[*of - (lan f)*]
by (*metis enat-ord-simps*(*2*) *ndropn-nfuse nfinite-nlength-enat ntaken-nfuse the-enat.simps zero-enat-def*)
have *4*: $\bigwedge x.\ (\lambda\ x.\ x \in somega\ (lan\ f))\ x \Longrightarrow$
$x \models ((f \wedge more) \wedge finite);(\lambda\ x.\ x \in somega\ (lan\ f))$

by (*metis 3 FMoreSem-var i0-less min-enat-simps*(*2*) *ntaken-nfuse ntaken-nlength somega.simps*)
have *5*: (λ *x*. *x*∈ *somega* (*lan f*)) *x*
by (*simp add*: *assms*)
show *?thesis* **using** *4 5 OmegaWeakCoinductSem*[*of* (λ *x*. *x*∈ *somega* (*lan f*)) *f*]
by (*simp add*: *schop-d-def*)
qed

lemma *somega-omega*:
x ∈ *somega* (*lan f*) ⟷ (*x* ⊨ f^{ω})
using *somega-omega-sem-1 somega-omega-sem-2* **by** *blast*

lemma *somega-omega-1*:
somega (*lan f*) = *lan*(*LIFT*(f^{ω}))
using *somega-omega* **by** *fastforce*

end

Chapter 4

Deep embedding of PITL

theory *PITL-Deep* **imports**
NELList-Extras

begin

This theory is a deep embedding of Propositional Interval Temporal Logic (PITL). It provides the syntax and semantics.

4.1 Syntax

4.1.1 Primitive formulae

datatype *pitl* =
false-d (*ifalse*)
| *atom-d nat* ((*atom* -))
| *iimp-d pitl pitl* ((- *iimp* -) [*26*,*25*] *25*)
| *next-d pitl* ((*next* -) [*88*] *87*)
| *chop-d pitl pitl* ((- *schop* -) [*84*,*84*] *83*)
| *schopstar-d pitl* ((*schopstar* -) [*85*] *85*)
| *omega-d pitl* ((*omega* -) [*85*] *85*)

4.1.2 state formula

fun *state-pitl* :: *pitl* ⇒ *bool*
where
state-pitl (*ifalse*) = *True*
| *state-pitl* (*atom p*) = *True*
| *state-pitl* (*f iimp g*) = ((*state-pitl f*) ∧ (*state-pitl g*))
| *state-pitl* (*next f*) = *False*
| *state-pitl* (*f schop g*) = *False*
| *state-pitl* (*schopstar f*) = *False*
| *state-pitl* (*omega f*) = *False*

4.1.3 Derived Boolean Operators

definition *inot-d* ((*inot* -) [*90*] *90*)

where
inot f ≡ *f iimp ifalse*

definition *itrue-d* (*itrue*)
where
itrue ≡ *inot ifalse*

definition *ior-d* ((- *ior* -) [*31*,*30*] *30*)
where
f ior g ≡ (*inot f*) *iimp g*

definition *iand-d* ((- *iand* -) [*36*,*35*] *35*)
where
f iand g ≡ *inot* (*inot f ior inot g*)

definition *ieqv-d* ((- *ieqv* -) [*21*,*20*] *20*)
where
f ieqv g ≡ ((*f iimp g*) *iand* (*g iimp f*))

4.1.4 more, empty, skip and wnext

definition *more-d* (*more*)
where
more ≡ *next itrue*

definition *empty-d* (*empty*)
where
empty ≡ *inot more*

definition *skip-d* (*skip*)
where
skip ≡ *next empty*

definition *wnext-d* ((*wnext* -) [*88*] *87*)
where
wnext f ≡ *inot* (*next* (*inot f*))

4.1.5 ifinite and inf

definition *finite-d* (*ifinite*)
where
ifinite ≡ *itrue schop empty*

definition *inf-d* (*inf*)
where
inf ≡ *inot ifinite*

4.1.6 weak chop

definition *chop-d* ((- ; -) [*84*,*84*] *83*)

where $f\ ;\ g \equiv (f\ schop\ g)\ ior\ (f\ iand\ inf)$

4.1.7 Box and Diamond Operators

definition *sometimes-d* ((*diamond* -) [*88*] *87*)
where
$diamond\ f \equiv itrue\ schop\ f$

definition *always-d* ((*box* -) [*88*] *87*)
where
$box\ f \equiv inot\ (diamond\ (inot\ f))$

4.1.8 Initial and Final Operators

definition *init-d* ((*init* -) [*88*] *87*)
where
$init\ f \equiv ((empty\ iand\ f)\ schop\ itrue)$

definition *fin-d* ((*fin* -) [*88*] *87*)
where
$fin\ f \equiv diamond\ (\ empty\ iand\ f)$

4.1.9 Big operations

definition *big-ior* :: $pitl\ list \Rightarrow\ pitl$
where $big\text{-}ior\ F = foldr\ (\lambda\ x\ y.\ x\ ior\ y)\ F\ ifalse$

definition *big-iand* :: $pitl\ list \Rightarrow\ pitl$
where $big\text{-}iand\ F = foldr\ (\lambda\ x\ y.\ x\ iand\ y)\ F\ itrue$

definition *state-pitl-list* :: $pitl\ list \Rightarrow bool$
where $state\text{-}pitl\text{-}list\ L = foldr\ (\lambda x\ y.\ state\text{-}pitl\ x \wedge y)\ L\ True$

4.2 Semantics

4.2.1 Intervals

type-synonym $interval =\ nat\ set\ nellist$

4.2.2 Semantics Primitive Operators

fun *semantics-pitl* :: [*interval*, *pitl*] ⇒ *bool* ((- ⊨ -) [*80*,*10*] *10*)
where
$(\sigma \models ifalse) = False$
| $(\sigma \models atom\ p) = (p \in (nfirst\ \sigma))$
| $(\sigma \models f\ iimp\ g) = ((\sigma \models f) \longrightarrow (\sigma \models g))$
| $(\sigma \models (next\ f)) = (1 \leq nlength\ \sigma\ \wedge ((ndropn\ 1\ \sigma) \models f))$
| $(\sigma \models f\ schop\ g) =$
$(\exists\, i.\ ((0 \leq i) \wedge (i \leq (nlength\ \sigma)) \wedge ((ntaken\ i\ \sigma) \models f) \wedge ((ndropn\ i\ \sigma) \models g))\ )$

| $(\sigma \models (schopstar\ f)) =$
$(\exists\ (l::\ nat\ nellist).$
$(nnth\ l\ 0) = 0 \wedge nfinite\ l \wedge nidx\ l\ \wedge$
$(enat\ (nlast\ l)) = (nlength\ \sigma) \wedge nfinite\ \sigma \wedge$
$(\forall\ (i::nat)\ .\ (\ (enat\ i) < (nlength\ l)) \longrightarrow$
$((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\ ) \models f)\ )$
)

| $(\sigma \models (omega\ f)) =$
$(\exists (l::\ nat\ nellist).$
$\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ (\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models f)\ )$
)

lemma *inot-defs* [*simp*]:
$(\sigma \models inot\ f\ ) = Not\ (\ \sigma \models f)$
by (*simp add*: *inot-d-def*)

lemma *ior-defs* [*simp*]:
$(\sigma \models f1\ ior\ f2) = (\ (\sigma \models f1) \vee (\sigma \models f2))$
by (*metis inot-defs ior-d-def semantics-pitl.simps*(*3*))

lemma *iand-defs* [*simp*]:
$(\sigma \models f1\ iand\ f2) = (\ (\sigma \models f1) \wedge (\sigma \models f2))$
by (*simp add*: *iand-d-def*)

lemma *ieqv-defs* [*simp*]:
$(\sigma \models f1\ ieqv\ f2) = (\ (\sigma \models f1) = (\sigma \models f2))$
by (*metis iand-defs ieqv-d-def semantics-pitl.simps*(*3*))

lemma *itrue-defs* [*simp*]:
$(\sigma \models itrue)$
by (*simp add*: *itrue-d-def*)

lemma *empty-defs* [*simp*]:
$(\sigma \models empty) = (\ (nlength\ \sigma) = 0)$
by (*auto simp add*: *empty-d-def more-d-def*)
(*metis co.enat.exhaust-sel enat-le-plus-same*(*2*) *plus-1-eSuc*(*2*))

lemma *state-pitl-defs*:
assumes *state-pitl w*
shows $(\sigma \models w) \longleftrightarrow ((NNil\ (nfirst\ \sigma)) \models w)$
using *assms*
proof (*induct w*)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)

then show *?case*
by (*metis nfirst-nfuse nfuse-leftneutral nlast-NNil semantics-pitl.simps*(*2*))
next
case (*iimp-d w1 w2*)
then show *?case* **by** *simp*
next
case (*next-d w*)
then show *?case* **by** *auto*
next
case (*chop-d w1 w2*)
then show *?case* **by** *auto*
next
case (*schopstar-d w*)
then show *?case* **by** *auto*
next
case (*omega-d w*)
then show *?case* **by** *auto*
qed

lemma *state-pitl-list-defs*:
state-pitl-list L $\longleftrightarrow$ ($\forall$ *i*<*length L. state-pitl* (*L*!*i*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *state-pitl-list-def* **by** *simp*
next
case (*Cons a L*)
then show *?case*
proof −
have *1*: *state-pitl-list* (*a* # *L*) $\longleftrightarrow$ *state-pitl a* $\wedge$ *state-pitl-list L*
unfolding *state-pitl-list-def* **by** *simp*
have *2*: ($\forall$ *i*<*length* (*a*#*L*). *state-pitl* ((*a*#*L*)!*i*)) $\longleftrightarrow$
state-pitl ((*a*#*L*)!*0*) $\wedge$
($\forall$ *i*. *0*<*i* $\wedge$ *i*<*length* (*a*#*L*)$\longrightarrow$ *state-pitl* ((*a*#*L*)!*i*))
by (*metis length-greater-0-conv list.discI zero-less-iff-neq-zero*)
have *3*: *state-pitl* ((*a*#*L*)!*0*) $\longleftrightarrow$ *state-pitl a*
by *simp*
have *4*: ($\forall$ *i*. *0*<*i* $\wedge$ *i*<*length* (*a*#*L*)$\longrightarrow$ *state-pitl* ((*a*#*L*)!*i*)) $\longleftrightarrow$
($\forall$ *i*<*length* (*L*). *state-pitl* (*L*!*i*))
by *auto*
show *?thesis*
using *1 2 3 4 local.Cons* **by** *presburger*
qed
qed

lemma *big-ior-defs* :
(*σ* $\models$ (*big-ior F*)) $\longleftrightarrow$ ($\exists$ *i*< *length F*. (*σ* $\models$ *F* ! *i*))
proof (*induct F*)
case *Nil*

then show *?case* **unfolding** *big-ior-def* **by** *simp*
next
case (*Cons a F*)
then show *?case*
proof −
have *1*: ($\sigma \models$ (*big-ior* (*a*#*F*))) $\longleftrightarrow$ ($\sigma \models$ *a ior* (*big-ior F*))
unfolding *big-ior-def* **by** *simp*
have *2*: ($\exists$ *i*<*length* (*a* # *F*). $\sigma \models$ (*a* # *F*) ! *i*) $\longleftrightarrow$
($\sigma \models$ (*a* # *F*) ! *0*) $\vee$
($\exists$ *i*. *0* < *i* $\wedge$ *i*<*length* (*a* # *F*) $\wedge$ ($\sigma \models$ (*a* # *F*) ! *i*))
by (*metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero*)
have *3*: ($\sigma \models$ (*a* # *F*) ! *0*) = ($\sigma \models$ *a*)
by *auto*
have *4*: ($\exists$ *i*. *0* < *i* $\wedge$ *i*<*length* (*a* # *F*) $\wedge$ ($\sigma \models$ (*a* # *F*) ! *i*)) $\longleftrightarrow$
($\exists$ *i*<*length* (*F*). ($\sigma \models$ (*F*) ! *i*))
by *auto*
(*metis Suc-le-eq diff-Suc-Suc diff-zero gr0-conv-Suc less-Suc-eq-le*)
show *?thesis*
using *1 2 4 local.Cons* **by** *force*
qed
qed

lemma *big-iand-defs* :
($\sigma \models$ (*big-iand F*)) $\longleftrightarrow$ ($\forall$ *i*< *length F*. ($\sigma \models$ *F* ! *i*))
proof (*induct F*)
case *Nil*
then show *?case* **unfolding** *big-iand-def* **by** *simp*
next
case (*Cons a F*)
then show *?case*
proof −
have *1*: ($\sigma \models$ (*big-iand* (*a*#*F*))) $\longleftrightarrow$ ($\sigma \models$ *a iand* (*big-iand F*))
unfolding *big-iand-def* **by** *simp*
have *2*: ($\forall$ *i*<*length* (*a* # *F*). $\sigma \models$ (*a* # *F*) ! *i*) $\longleftrightarrow$
($\sigma \models$ (*a* # *F*) ! *0*) $\wedge$
($\forall$ *i*. *0* < *i* $\wedge$ *i*<*length* (*a* # *F*) $\longrightarrow$ ($\sigma \models$ (*a* # *F*) ! *i*))
by (*metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero*)
have *3*: ($\sigma \models$ (*a* # *F*) ! *0*) = ($\sigma \models$ *a*)
by *auto*
have *4*: ($\forall$ *i*. *0* < *i* $\wedge$ *i*<*length* (*a* # *F*) $\longrightarrow$ ($\sigma \models$ (*a* # *F*) ! *i*)) $\longleftrightarrow$
($\forall$ *i*<*length* (*F*). ($\sigma \models$ (*F*) ! *i*))
by *auto*
show *?thesis*
using *1 2 4 local.Cons* **by** *force*
qed
qed

lemma *big-ior-map-iand-zip* :

assumes *length F =length G*
shows ($\sigma \models$ *big-ior* (*map2 iand-d F G*)) $\longleftrightarrow$
($\exists\, i <$ *length F*. ($\sigma \models F$! *i iand G* ! *i*))
using *assms*
proof (*induct F arbitrary*: *G*)
case *Nil*
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*Cons a F*)
then show *?case*
proof (*cases G*=[])
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** (*auto simp add*: *big-ior-defs*)
qed
qed

lemma *big-iand-map-iimp-zip* :
assumes *length F =length G*
shows ($\sigma \models$ *big-iand* (*map2 iimp-d F G*)) $\longleftrightarrow$
($\forall\, i <$ *length F*. ($\sigma \models F$! *i iimp G* ! *i*))
using *assms*
proof (*induct F arbitrary*: *G*)
case *Nil*
then show *?case* **by** (*simp add*: *big-iand-defs*)
next
case (*Cons a F*)
then show *?case*
proof (*cases G*=[])
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** (*auto simp add*: *big-iand-defs*)
qed
qed

lemma *map2-iand-defs* :
assumes *length F = length G*
shows ($\exists\, i$<*length G*. $\sigma \models$ *map2 iand-d F G* ! *i*) $\longleftrightarrow$
($\exists\, i$<*length G*. ($\sigma \models F$!*i*) $\wedge$ ($\sigma \models G$! *i*))
using *assms*
proof (*induct G arbitrary*: *F*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a G*)

then show *?case*
proof (*cases F=[]*)
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** *auto*
qed
qed

lemma *map2-iand-inot-defs* :
assumes *length F = length G*
shows ($\exists$ *i*<*length G*. $\sigma \models$ *map2* ($\lambda x\ y.\ x$ *iand inot* y) *F G* ! *i*) $\longleftrightarrow$
($\exists$ *i*<*length G*. ($\sigma \models F!\ i$) $\wedge$ $\neg$($\sigma \models$ *G*!*i*))
using *assms*
proof (*induct G arbitrary*: *F*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a G*)
then show *?case*
proof (*cases F=[]*)
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** *auto*
qed
qed

4.2.3 Validity

definition *valid* :: *pitl* $\Rightarrow$ *bool* (($\vdash$ -) 5)
where ($\vdash f$) = ($\forall\ \sigma.\ (\sigma \models f)$)

lemma *itl-valid* [*simp*] :
($\vdash f$) = ($\forall\ \sigma.\ (\sigma \models f)$)
by (*simp add*: *valid-def*)

lemma *itl-ieq*:
($\vdash f$ *ieqv* g) = ($\forall\ \sigma.\ (\sigma \models f) = (\sigma \models g)$)
by (*auto simp add*: *ieqv-d-def iand-d-def ior-d-def inot-d-def*)

lemma *big-ior-iimp-subset-eq*:
assumes *set L1* $\subseteq$ *set L2*
shows $\vdash$ *big-ior L1 iimp big-ior L2*
using *assms*
proof (*induct rule*: *list-induct2'*)
case *1*
then show *?case* **by** *simp*

next
case (*2 x xs*)
then show *?case* **by** *simp*
next
case (*3 y ys*)
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*4 x xs y ys*)
then show *?case*
proof (*cases x=y*)
case *True*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-ior-defs*)
(*metis diff-Suc-1 in-set-conv-nth length-Cons less-Suc-eq-0-disj list.simps*(*15*) *nth-Cons' subsetD*)
next
case *False*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-ior-defs*)
(*metis 4.prems in-set-conv-nth length-Cons subsetD*)
qed
qed

lemma *big-ior-ieqv-set-eq*:
assumes *set L1 = set L2*
shows ⊢ *big-ior L1 ieqv big-ior L2*
using *assms*
by (*metis big-ior-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-ior-ieqv-remdups*:
shows ⊢ *big-ior L ieqv big-ior* (*remdups L*)
by (*metis big-ior-ieqv-set-eq set-remdups*)

lemma *big-iand-iimp-subset-eq*:
assumes *set L1* ⊆ *set L2*
shows ⊢ *big-iand L2 iimp big-iand L1*
using *assms*
proof (*induct rule*: *list-induct2'*)
case *1*
then show *?case* **by** *simp*
next
case (*2 x xs*)
then show *?case* **by** (*simp add*: *big-iand-defs*)
next
case (*3 y ys*)
then show *?case* **by** *simp*
next
case (*4 x xs y ys*)
then show *?case*
proof (*cases x=y*)
case *True*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-iand-defs*)

(*metis* (*mono-tags*, *lifting*) *4.prems all-nth-imp-all-set length-Cons list-ball-nth subsetD*)
next
case *False*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-iand-defs*)
(*metis 4.prems add.right-neutral add-Suc-right in-set-conv-nth list.size*(*4*) *subsetD*)
qed
qed

lemma *big-iand-ieqv-set-eq*:
assumes *set L1* = *set L2*
shows ⊢ *big-iand L1 ieqv big-iand L2*
using *assms*
by (*metis big-iand-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-iand-ieqv-remdups*:
⊢ *big-iand L ieqv big-iand* (*remdups L*)
by (*metis big-iand-ieqv-set-eq set-remdups*)

lemma *big-ior-empty*:
(⊢ *big-ior* []) ⟷ (⊢ *ifalse*)
by (*auto simp add*: *big-ior-defs*)

lemma *big-ior-Cons*:
(⊢ *big-ior* (*f* # *L*)) ⟷ (⊢ *f ior* (*big-ior L*))
proof −
have *1*: (⊢ *big-ior* (*f* # *L*)) ⟹ (⊢ *f ior* (*big-ior L*))
by (*auto simp add*: *big-ior-defs*)
(*metis diff-Suc-1 gr0-conv-Suc less-Suc-eq less-imp-diff-less nth-Cons'*)
have *2*: (⊢ *f ior* (*big-ior L*)) ⟹ (⊢ *big-ior* (*f* # *L*))
by (*auto simp add*: *big-ior-defs*) *fastforce*
show *?thesis* **using** *1 2* **by** *auto*
qed

lemma *big-ior-Cons-alt*:
⊢ *big-ior* (*f* # *L*) *ieqv f ior big-ior L*
unfolding *big-ior-def*
by *auto*

lemma *big-ior-1*:
assumes ⋀*i*. *i*< *length F* ⟹ (⊢ *F* ! *i ieqv G* ! *i*)
length F = *length G*
shows ⊢ (*big-ior F*) *ieqv* (*big-ior G*)
using *assms*
by (*auto simp add*: *big-ior-defs*)

lemma *big-iand-1*:
assumes $\bigwedge i.\ i < length\ F \Longrightarrow\ (\vdash F\ !\ i\ ieqv\ G\ !\ i)$
length F = length G
shows $\vdash$ (*big-iand F*) *ieqv* (*big-iand G*)
using *assms*
by (*auto simp add*: *big-iand-defs*)

lemma *big-ior-big-iand-1*:
$\vdash$ (*inot* (*big-ior F*)) *ieqv* (*big-iand* (*map* ($\lambda x.$ *inot x*) *F*))
by (*auto simp add*: *big-ior-defs big-iand-defs*)

lemma *big-ior-big-iand-2*:
$\vdash$ (*inot* (*big-iand F*)) *ieqv* (*big-ior* (*map* ($\lambda x.$ *inot x*) *F*))
by (*auto simp add*: *big-ior-defs big-iand-defs*)

lemma *big-ior-append*:
($\vdash$ (*big-ior* (*L1@L2*)) *ieqv* (*big-ior L1*) *ior* (*big-ior L2*))
proof (*induct L1 arbitrary*: *L2*)
case *Nil*
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*Cons a L1a*)
then show *?case*
proof −
have *1*: $\vdash$ *big-ior* ((*a* # *L1a*) @ *L2*) *ieqv a ior* ((*big-ior* (*L1a* @ *L2*)))
using *big-ior-Cons-alt* **by** *auto*
have *2*: $\vdash$ *big-ior* (*L1a* @ *L2*) *ieqv big-ior L1a ior big-ior L2*
using *local.Cons* **by** *blast*
have *3*: $\vdash$ *a ior* ((*big-ior* (*L1a* @ *L2*))) *ieqv a ior* (*big-ior L1a ior big-ior L2*)
using *1 2* **by** *auto*
have *4*: $\vdash$ *big-ior* (*a* # *L1a*) *ieqv a ior* (*big-ior L1a*)
using *big-ior-Cons-alt* **by** *blast*
have *5*: $\vdash$ *a ior* (*big-ior L1a ior big-ior L2*) *ieqv big-ior* (*a* # *L1a*) *ior big-ior L2*
using *4* **by** *auto*
show *?thesis* **using** *1 3 5* **by** *force*
qed
qed

lemma *state-pitl-list-append*:
shows *state-pitl-list* (*L@L1*) $\longleftrightarrow$ *state-pitl-list L* $\wedge$ *state-pitl-list L1*
proof (*induct L arbitrary*: *L1*)
case *Nil*
then show *?case*
by (*simp add*: *state-pitl-list-defs*)
next
case (*Cons a L*)
then show *?case*
proof −

```
    have 1: state-pitl-list ((a # L) @ L1) ⟷ state-pitl a ∧ state-pitl-list (L @ L1)
      unfolding state-pitl-list-def by auto
    have 2: state-pitl-list (L @ L1) ⟷ state-pitl-list L ∧ state-pitl-list L1
      by (simp add: local.Cons)
    have 3: state-pitl a ∧ state-pitl-list L ∧ state-pitl-list L1 ⟷
          (state-pitl-list (a # L) ∧ state-pitl-list L1)
      by (simp add: state-pitl-list-def)
    show ?thesis using 1 3 local.Cons by auto
   qed
qed


lemma map2-map:
 assumes length L1 = length L2
 shows    map2 f (map g L1) (map h L2) = map2 (λx y. f (g x) (h y)) L1 L2
using assms
proof (induct L1 L2 rule:list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case by auto
qed


lemma map2-map-a:
assumes length L1 = length L2
shows (map2 (λx y. x iand next y) (map (λx1. x iand x1) L1) (map (λy1. y ior y1) L2)) =
      (map2 (λx1 y1. (x iand x1) iand next ( y ior y1)) L1 L2)
using assms map2-map by blast

lemma map2-map-a-a:
assumes length L1 = length L2
shows (map2 (λx y. x iand next y)
            (map (λx1. x iand x1) L1)
            (map (λy1. (if y = ifalse then y1 else y ior y1)) L2)) =
      (map2 (λx1 y1. (x iand x1) iand next ( (if y = ifalse then y1 else y ior y1))) L1 L2)
using assms map2-map by blast


lemma map2-map-b:
assumes length L1 = length L2
shows (map2 (λx y. x iand next y) (map (λx1. (inot x) iand x1) L1) L2) =
      (map2 (λx1 y1. ((inot x) iand x1) iand next  y1) L1 L2)
using assms map2-map[of L1 L2 (λx y. x iand next y)  (λx1. (inot x) iand x1) id]
by auto


lemma big-ior-big-ior-map2:
assumes length L1 = length L2
```

$\vdash$ *big-ior L1*
shows $\vdash$ *big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
using *assms*
proof (*induct L1 L2 rule*:*list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
by (*metis fst-def list.size*(*4*) *map-eq-conv map-fst-zip*)
qed

lemma *big-ior-split-off*:
assumes *length L1* = *length L2*
($\vdash$ *big-ior L1*)
shows $\vdash$ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*y ior y1*))) *L1 L2*)) *ior*
(*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*x iand next y*) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*) *L1 L2*))
proof −
have *1*: $\vdash$ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*y ior y1*))) *L1 L2*)) *ieqv*
(*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand* (*next* (*y ior y1*))) *L1 L2*)))
by (*auto simp add*: *big-ior-defs*)
have *2*: $\vdash$ (*big-ior* (*map2* (λ*x1 y1*. *x1 iand* (*next* (*y ior y1*))) *L1 L2*)) *ieqv*
((*next y iand* (*big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)))
ior
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))
by (*auto simp add*: *big-ior-defs*)
have *3*: $\vdash$ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*y ior y1*))) *L1 L2*)) *ieqv*
(*x iand* (*next y iand big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
ior
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))))
by (*auto simp add*: *big-ior-defs*)
have *4*: $\vdash$ *big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
using *assms big-ior-big-ior-map2* **by** *blast*
have *5*: $\vdash$ (*x iand* (*next y iand big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
ior
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))) *ieqv*
(*x iand next y*) *ior* (*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))
using *4* **by** *auto*
have *6*: $\vdash$ (*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand* (*next* (*y1*))) *L1 L2*)))
by (*auto simp add*: *big-ior-defs*)
have *7*: $\vdash$ (*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ior*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))
by (*auto simp add*: *big-ior-defs*)
have *8* : $\vdash$ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*y ior y1*))) *L1 L2*)) *ior*
(*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*

```
          (x iand next y) ior
          (x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 ))) ior
          (inot x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))
       using 5 6 3 by auto
  show ?thesis using 7 8 by auto
qed

lemma big-ior-split-off-a:
assumes length L1 = length L2
      (⊢ big-ior L1)
shows ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )) ior
       (big-ior  (map2 (λx1 y1. (inot x iand  x1) iand next  y1)      L1 L2 )) ieqv
       (x iand next y) ior
       (big-ior  (map2 (λx y.  x iand next  y)  L1 L2 ))
proof −
 have 1: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )) ieqv
         (x iand (big-ior  (map2 (λx1 y1.  x1 iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )))
      by (auto simp add: big-ior-defs)
 have 2: ⊢ (big-ior  (map2 (λx1 y1.  x1 iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )) ieqv
          ((next y iand (big-ior  (map2 (λx1 y1.  x1 ) L1 L2 )))
          ior
          (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))
   proof (cases y=ifalse)
   case True
   then show ?thesis by simp
   next
   case False
   then show ?thesis by (auto simp add: big-ior-defs)
   qed
 have 3: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand ( next (if y = ifalse then y1 else y ior y1))) L1 L2 )) ieqv
          (x iand  (next y iand big-ior  (map2 (λx1 y1.  x1) L1 L2 )
           ior
          (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 ))))
    proof (cases y=ifalse)
    case True
    then show ?thesis by (auto simp add: big-ior-defs)
    next
    case False
    then show ?thesis by (auto simp add: big-ior-defs)
    qed
  have 4: ⊢ big-ior  (map2 (λx1 y1.  x1 ) L1 L2 )
       using assms big-ior-big-ior-map2 by blast
  have 5: ⊢ (x iand  (next y iand big-ior  (map2 (λx1 y1.  x1 )  L1 L2 )
           ior
           (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))) ieqv
          (x iand next y) ior (x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))
      using 4 by auto
```

have *6*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand* (*next* (*y1*))) *L1 L2*)))
by (*auto simp add*: *big-ior-defs*)
have *7*: ⊢ (*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ior*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))
by (*auto simp add*: *big-ior-defs*)
have *8* : ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*if y = ifalse then y1 else y ior y1*))) *L1 L2*)) *ior*
(*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*x iand next y*) *ior*
(*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ior*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))
using *5 6 3* **by** *auto*
show *?thesis* **using** *7 8* **by** *auto*
qed

lemma *state-pitl-list-map*:
assumes *state-pitl f*
state-pitl-list L
shows *state-pitl-list* (*map* (λ *x*. *f iand x*) *L*)
using *assms*
proof (*induct L*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L*)
then show *?case*
proof −
have *1*: *state-pitl-list* (*map* (λ*x*. *f iand x*) (*a* # *L*)) ⟷
state-pitl (*f iand a*) ∧ *state-pitl-list* ((*map* (λ*x*. *f iand x*) *L*))
unfolding *state-pitl-list-def* **by** *auto*
have *2*: *state-pitl-list* (*a* # *L*) ⟷ *state-pitl a* ∧ *state-pitl-list L*
unfolding *state-pitl-list-def* **by** *auto*
have *3*: *state-pitl* (*f iand a*)
unfolding *iand-d-def inot-d-def ior-d-def*
using *2 Cons.prems*(*2*) *assms*(*1*) **by** *auto*
show *?thesis*
using *1 2 3 Cons.hyps Cons.prems*(*2*) *assms*(*1*) **by** *blast*
qed
qed

lemma *state-pitl-list-map-alt*:
assumes *state-pitl f*
state-pitl-list L
shows *state-pitl-list* (*map* (λ *x*. *x iand f*) *L*)
using *assms*
proof (*induct L*)
case *Nil*

then show *?case* **by** *simp*
next
case (*Cons a L*)
then show *?case*
proof −
have *1*: *state-pitl-list* (*map* (λ*x*. *x iand f*) (*a* # *L*)) ⟷
state-pitl (*a iand f*) ∧ *state-pitl-list* ((*map* (λ*x*. *x iand f*) *L*))
unfolding *state-pitl-list-def* **by** *auto*
have *2*: *state-pitl-list* (*a* # *L*) ⟷ *state-pitl a* ∧ *state-pitl-list L*
unfolding *state-pitl-list-def* **by** *auto*
have *3*: *state-pitl* (*a iand f*)
unfolding *iand-d-def inot-d-def ior-d-def*
using *2 Cons.prems*(*2*) *assms*(*1*) **by** *auto*
show *?thesis*
using *1 2 3 Cons.hyps Cons.prems*(*2*) *assms*(*1*) **by** *blast*
qed
qed

lemma *map-fst-filter-sat*:
assumes *length L1* = *length L2*
shows *map fst* (*filter* (λ (*x*,*y*). (∃ σ. σ ⊨ *x*)) (*zip L1 L2*)) =
filter (λ*x*. (∃ σ. σ ⊨ *x*)) *L1*
using *assms*
proof (*induct L1 L2 rule*: *list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
proof (*cases xs* = [] ∧ *ys* = [])
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then show *?thesis*
using *Cons* **by** *auto*
qed
qed

lemma *big-ior-ieqv-filter-sat*:
⊢ *big-ior L ieqv big-ior* (*filter* (λ*x*. (∃ σ. σ ⊨ *x*)) *L*)
proof (*induct L*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L*)
then show *?case*
using *big-ior-Cons-alt* **by** *auto*
qed

lemma *total-filter-sat*:
assumes $\vdash$ *big-ior L*
shows $\vdash$ *big-ior* (*filter* ($\lambda x.\ (\exists\ \sigma.\ \sigma \models x)$) *L*)
using *assms*
using *big-ior-ieqv-filter-sat* **by** *auto*

lemma *big-ior-filter-sat-eqv*:
assumes *length L1* = *length L2*
shows $\vdash$ *big-ior* (*map* (λ (*x*,*y*) . *x iand* (*next y*))
(*filter* (λ (*x*,*y*). ($\exists$ σ. $\sigma \models x$)) (*zip L1 L2*)))
ieqv
big-ior (*map2* (λ *x y*. *x iand* (*next y*)) *L1 L2*)
using *assms*
proof (*induct L1 L2 rule*: *list-induct2*)
case *Nil*
then show *?case* **by** (*simp add*: *big-ior-def*)
next
case (*Cons x xs y ys*)
then show *?case*
proof (*cases xs* = [] $\wedge$ *ys* = [])
case *True*
then show *?thesis* **using** *Cons*
unfolding *big-ior-def big-iand-def*
by *simp*
next
case *False*
then show *?thesis*
proof −
have *1*: *length xs* = *length ys*
by (*simp add*: *Cons.hyps*(*1*))
have *2*: $\vdash$ *big-ior* (*map* (λ(*x*, *y*). *x iand next y*)
(*filter* (λ(*x*, *y*). $\exists\sigma.\ \sigma \models x$)
(*zip xs ys*)))
ieqv big-ior (*map2* (λ*x y*. *x iand next y*) *xs ys*)
using *Cons.hyps*(*2*) *False* **by** *fastforce*
show *?thesis*
using *2 big-ior-Cons-alt* **by** *auto*
qed
qed
qed

lemma *big-ior-rev-ieqv*:
$\vdash$ *big-ior L ieqv big-ior* (*rev L*)
by (*auto simp add*: *big-ior-defs*)
(*metis in-set-conv-nth length-rev set-rev*,
metis diff-less length-greater-0-conv less-nat-zero-code list.size(*3*) *rev-nth zero-less-Suc*)

lemma *big-ior-map2-append*:
assumes *length L1= length N1*
length L2= length N2
shows $\vdash$ (*big-ior* (*map2* (λ *x y . x iand* (*next y*)) (*L1@L2*) (*N1@N2*))) *ieqv*
(*big-ior* (*map2* (λ *x y . x iand* (*next y*)) *L1 N1*)) *ior*
(*big-ior* (*map2* (λ *x y . x iand* (*next y*)) *L2 N2*))
using *assms*
proof (*induct L1 N1 arbitrary*: *L2 N2 rule*:*list-induct2*)
case *Nil*
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*Cons x xs y ys*)
then show *?case*
by (*metis Cons.hyps*(*1*) *big-ior-append length-Cons map-append zip-append*)

qed

lemma *map2-iand-next-defs* :
assumes *length F = length G*
shows ($\exists$ *i*<*length G*. $\sigma \models$ *map2* (λ*x y. x iand* (*next y*)) *F G* ! *i*) $\longleftrightarrow$
($\exists$ *i*<*length G*. ($\sigma \models$ *F* !*i*) $\wedge$ ($\sigma \models$ (*next* (*G* ! *i*))))
using *assms*
proof (*induct G arbitrary*: *F*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a G*)
then show *?case*
proof (*cases F*=[])
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** *auto*
qed
qed

lemma *map2-iand-inot-next-defs* :
assumes *length F = length G*
0<*nlength* σ
shows ($\exists$ *i*<*length G*. $\sigma \models$ *map2* (λ*x y. x iand* (*next* (*inot y*))) *F G* ! *i*) $\longleftrightarrow$
($\exists$ *i*<*length G*. ($\sigma \models$ *F*! *i*) $\wedge$ $\neg$($\sigma \models$ (*next* (*G*!*i*))))
using *assms*
proof (*induct G arbitrary*: *F*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a G*)
then show *?case*

```
  proof (cases F=[])
  case True
  then show ?thesis using Cons by simp
  next
  case False
  then show ?thesis
   proof −
    obtain b F1 where 1: F = b#F1
     using False by (meson neq-Nil-conv)
    have 2:  length F1 = length G
       using 1 Cons.prems by auto
    have 3: (∃ i<length G. σ ⊨ map2 (λx y. x iand next inot y) F1 G ! i) =
          (∃ i<length G. (σ ⊨ F1 ! i) ∧ ¬ (σ ⊨ next G ! i))
     using 2 Cons.hyps assms(2) by presburger
    have 4: (∃ i<length (a # G). σ ⊨ map2 (λx y. x iand next inot y) F (a # G) ! i) ⟷
          (σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! 0) ∨
          (∃ i. 0 < i ∧ i<length (a # G) ∧ ( σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! i))
      using 1  by (metis gr-zeroI length-greater-0-conv list.discI)
    have 5: (σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! 0) ⟷
          (σ ⊨ b iand next inot a)
     by auto
    have 6: (∃ i. 0 < i ∧ i<length (a # G) ∧ ( σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) !
i)) ⟷
          (∃ i<length ( G).  ( σ ⊨ map2 (λx y. x iand next inot y) (F1) (G) ! i))
      by auto
        (metis Suc-less-eq Suc-pred)
    have 7: (∃ i<length (a # G). (σ ⊨ (b#F1) ! i) ∧ ¬ (σ ⊨ next (a # G) ! i))  ⟷
          ((σ ⊨ (b#F1) ! 0) ∧ ¬ (σ ⊨ next (a # G) ! 0)) ∨
          (∃ i. 0< i ∧ i<length (a # G) ∧ (σ ⊨ (b#F1) ! i) ∧ ¬ (σ ⊨ next (a # G) ! i))
      by (metis gr-zeroI length-greater-0-conv list.discI)
    have 8: ((σ ⊨ (b#F1) ! 0) ∧ ¬ (σ ⊨ next (a # G) ! 0)) ⟷
          (σ ⊨ b iand next inot a)
      by simp
       (metis One-nat-def Suc-ile-eq assms(2) enat-defs(1) enat-defs(2))
    have 9: (∃ i. 0< i ∧ i<length (a # G) ∧ (σ ⊨ (b#F1) ! i) ∧ ¬ (σ ⊨ next (a # G) ! i)) ⟷
          (∃ i<length ( G). (σ ⊨ (F1) ! i) ∧ ¬ (σ ⊨ next ( G) ! i))
       by auto
        (metis Suc-diff-Suc Suc-less-eq diff-zero, metis Suc-less-eq Suc-pred)
    show ?thesis
    using 1 3 4 5 6 7 8 9 by presburger
   qed
  qed
qed

lemma iand-big-ior-dist:
assumes length G-now = length G-next
shows ⊢ F-e iand big-ior (map2 (λ x y. x iand next y) G-now G-next) ieqv
       big-ior (map2 (λ x y. x iand next y)
                     (map (λx. x iand F-e) G-now) G-next)
using assms by (auto simp add: big-ior-defs)
```

lemma *big-ior-map2-defs*:
assumes *length L1 = length L2*
shows $(\sigma \models (\textit{big-ior}\ (\textit{map2}\ (\lambda x\ y.\ f\ x\ y)\ L1\ L2))) \longleftrightarrow$
$(\exists\, i{<}\textit{length}\ L1.\ (\sigma \models f\ (L1!i)\ (L2!i)))$
using *assms*
proof (*induct L1 L2 rule*: *list-induct2*)
case *Nil*
then show *?case* **unfolding** *big-ior-def* **by** *auto*
next
case (*Cons x xs y ys*)
then show *?case* **unfolding** *big-ior-def* **using** *less-Suc-eq-0-disj* **by** *auto*
qed

lemma *big-ior-map2-schop-dis*:
assumes *length L1 = length L2*
shows $\vdash (\textit{big-ior}\ (\textit{map2}\ (\lambda x\ y.\ f\ x\ y)\ L1\ L2))\ \textit{schop}\ g\ \textit{ieqv}$
$(\textit{big-ior}\ (\textit{map2}\ (\lambda x\ y.\ (f\ x\ y)\ \textit{schop}\ g)\ L1\ L2))$
using *assms*
by (*auto simp add*: *big-ior-map2-defs*)

end

theory *PITL-Deep-GNF* **imports**
PITL-Deep

begin

This theory proves that all PITL formulae can be written in Guarded Normal Form (GNF) [5, 2].

4.3 Guarded Normal Form

4.3.1 Definitions

definition *mutex*
where $\textit{mutex}\ F =$
$(\forall\ k1\ k2.\ 0 \le k1\ \wedge k1 < k2\ \wedge k2 < \textit{length}\ F \longrightarrow (\vdash \textit{inot}\ ((F!\ k1)\ \textit{iand}\ (F!\ k2))))$

definition *mutex-single*
where $\textit{mutex-single}\ f\ F =$
$(\forall\ k2.\ 0 \le k2\ \wedge k2 < \textit{length}\ F \longrightarrow (\vdash \textit{inot}\ (f\ \textit{iand}\ (F!\ k2))))$

definition *add-to-now-single*
where
add-to-now-single f L = ((*map* (λ*x*. (*f iand x*)) *L*)@
(*map* (λ*x*. (*inot f iand x*)) *L*))

definition *add-to-next-single*
where
add-to-next-single f L = ((*map* (λ*x*. (*f ior x*)) *L*)@ *L*)

definition *add-to-single*
where
add-to-single f1 f2 L1 L2 =
filter (λ (*x*,*y*). (∃ σ. σ ⊨ *x*)) (*zip* (*add-to-now-single f1 L1*) (*add-to-next-single f2 L2*))

definition *add-to-next-single-a*
where
add-to-next-single-a f L = ((*map* (λ*x*. ((*if f* = *ifalse then x else f ior x*))) *L*)@ *L*)

definition *add-to-now*
where *add-to-now L1* = *foldr* (*add-to-now-single*) *L1* [*itrue*]

definition *add-to-next*
where *add-to-next L1* = *foldr* (*add-to-next-single*) *L1* [*ifalse*]

definition *add-to-next-a*
where *add-to-next-a L1* = *foldr* (*add-to-next-single-a*) *L1* [*ifalse*]

definition *add-to*
where *add-to L1 L2* = *filter* (λ (*x*,*y*). (∃σ . σ ⊨ *x*)) (*zip* (*add-to-now L1*) (*add-to-next L1*))

fun *GNF-empty* :: *pitl* ⇒ *pitl*
where
GNF-empty (*ifalse*) = *ifalse*
| *GNF-empty* (*atom p*) = (*atom p*)
| *GNF-empty* (*f iimp g*) = ((*inot* (*GNF-empty f*)) *ior* (*GNF-empty g*))
| *GNF-empty* (*next f*) = (*ifalse*)
| *GNF-empty* (*f schop g*) = ((*GNF-empty f*) *iand* (*GNF-empty g*))
| *GNF-empty* (*schopstar f*) = *itrue*
| *GNF-empty* (*omega f*) = *ifalse*

fun *GNF-state* :: *pitl* ⇒ *pitl list*
where
GNF-state (*ifalse*) = [*itrue*]

```
|  GNF-state (atom p)       = [(atom p)]
|  GNF-state (f iimp g)     = ( ((add-to-now (GNF-state f)    )) @ (GNF-state g))
|  GNF-state (next f)       = [itrue]
|  GNF-state (f schop g)    = ( (GNF-state f) @ (map (λx. x iand (GNF-empty f)) (GNF-state g)))
|  GNF-state (schopstar f) = (GNF-state f)
|  GNF-state (omega f)      = (GNF-state f)


fun GNF-next :: pitl ⇒ pitl list
where
   GNF-next (ifalse)       = [ifalse]
|  GNF-next (atom p)       = [itrue]
|  GNF-next (f iimp g)     = ( (map (λx. (inot x)) (add-to-next (GNF-next f) )) @ (GNF-next g))
|  GNF-next (next f)       = [f]
|  GNF-next (f schop g)    = ( (map (λx. x schop g) (GNF-next f)) @  (GNF-next g))
|  GNF-next (schopstar f) = (map (λx. x schop (schopstar f)) (GNF-next f))
|  GNF-next (omega f)      = (map (λx. x schop (omega f)) (GNF-next f))


fun GNF-next-a :: pitl ⇒ pitl list
where
   GNF-next-a (ifalse)       = [ifalse]
|  GNF-next-a (atom p)       = [itrue]
|  GNF-next-a (f iimp g)     = ( (map (λx. (inot x)) (add-to-next-a (GNF-next-a f) )) @ (GNF-next-a g))
|  GNF-next-a (next f)       = [f]
|  GNF-next-a (f schop g)    = ( (map (λx. x schop g) (GNF-next-a f)) @  (GNF-next-a g))
|  GNF-next-a (schopstar f) = (map (λx. x schop (schopstar f)) (GNF-next-a f))
|  GNF-next-a (omega f)      = (map (λx. x schop (omega f)) (GNF-next-a f))



definition full-system
 where full-system F = ((⊢ big-ior F) ∧ mutex F)


definition GNF-cons
 where GNF-cons ae a A =
       ((ae iand empty) ior (big-ior (map2 (λ x y. x iand (next y))  a A  )))

definition GNF
 where GNF f =
      (GNF-cons (GNF-empty f) (GNF-state f) (GNF-next f))


definition GNF-prop
 where GNF-prop X =
      (∃ ae a A.
        (state-pitl ae) ∧ (state-pitl-list a) ∧
        full-system a ∧ length a = length A ∧
        (⊢ X ieqv GNF-cons ae a A)
```

)

definition *GNF-det*
where *GNF-det f* =
(*GNF-cons* (*GNF-empty f*)
(*foldr* (*add-to-now-single*) (*GNF-state f*) [*itrue*])
(*foldr* (*add-to-next-single*) (*GNF-next f*) [*ifalse*])
)

4.3.2 Mutex lemmas

lemma *mutex-single-empty*:
shows *mutex-single f* []
unfolding *mutex-single-def* **by** *auto*

lemma *mutex-single-Cons*:
shows *mutex-single f* (*f1* # *F*) ⟷
(⊢ *inot* (*f iand f1*)) ∧ *mutex-single f F*
unfolding *mutex-single-def* **using** *less-Suc-eq-0-disj* **by** *auto*

lemma *mutex-single-append*:
shows *mutex-single f* (*F1*@ *F2*) ⟷ *mutex-single f F1* ∧ *mutex-single f F2*
proof (*induct F1 arbitrary*: *F2*)
case *Nil*
then show *?case* **unfolding** *mutex-single-def* **by** *simp*
next
case (*Cons a F1*)
then show *?case*
proof −
have *1*: *mutex-single f* ((*a* # *F1*) @ *F2*) ⟷
(⊢ *inot* (*f iand a*)) ∧ *mutex-single f* (*F1* @*F2*)
by (*simp add*: *mutex-single-Cons*)
have *2*: *mutex-single f* (*F1* @*F2*) ⟷ *mutex-single f F1* ∧ *mutex-single f F2*
using *Cons* **by** *blast*
have *3*: (⊢ *inot* (*f iand a*)) ∧ (*mutex-single f F1* ∧ *mutex-single f F2*) ⟷
(*mutex-single f* (*a* # *F1*) ∧ *mutex-single f F2*)
using *mutex-single-Cons* **by** *blast*
show *?thesis*
using *1 2 3* **by** *presburger*
qed
qed

lemma *mutex-empty*:
shows *mutex* []
unfolding *mutex-def* **by** *auto*

lemma *mutex-Cons*:
shows *mutex* (*f* # *F1*) ⟷ *mutex-single f F1* ∧ *mutex F1*
unfolding *mutex-def mutex-single-def*
using *less-Suc-eq-0-disj*
apply *auto*
apply (*metis diff-Suc-1′ nth-Cons′*)
by (*metis diff-Suc-1′ nth-Cons-Suc*)

lemma *mutex-append*:
shows *mutex* (*F1* @ *F2*) ⟷
mutex F1 ∧ *foldr* (λ*x y*. (*mutex-single x F2*) ∧ *y*) *F1 True* ∧ *mutex F2*
proof (*induct F1 arbitrary*: *F2*)
case *Nil*
then show *?case*
by (*simp add*: *mutex-empty*)
next
case (*Cons a F1*)
then show *?case*
proof −
have *1*: *mutex* ((*a* # *F1*) @ *F2*) ⟷
mutex-single a F1 ∧ *mutex-single a F2* ∧ *mutex* (*F1*@*F2*)
by (*simp add*: *mutex-Cons mutex-single-append*)
have *2*: *foldr* (λ*x*. (∧) (*mutex-single x F2*)) (*a* # *F1*) *True* ⟷
mutex-single a F2 ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) (*F1*) *True*
by *simp*
have *3*: *mutex* (*F1* @ *F2*) ⟷ (*mutex F1* ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) *F1 True* ∧ *mutex F2*)
using *Cons* **by** *blast*
have *4*: *mutex-single a F1* ∧ *mutex-single a F2* ∧ *mutex* (*F1*@*F2*) ⟷
mutex-single a F1 ∧ *mutex-single a F2* ∧
(*mutex F1* ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) *F1 True* ∧ *mutex F2*)
using *1 3* **by** *auto*
have *5*: *mutex-single a F1* ∧ *mutex-single a F2* ∧
(*mutex F1* ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) *F1 True* ∧ *mutex F2*) ⟷
(*mutex* (*a* # *F1*) ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) (*a* # *F1*) *True* ∧ *mutex F2*)
using *2 mutex-Cons* **by** *blast*
show *?thesis*
using *1 4 5* **by** *presburger*
qed
qed

lemma *unfold-mutex-single*:
foldr (λ*x y*. (*mutex-single x F2*) ∧ *y*) *F1 True* ⟷
(∀ *i*. *0* ≤ *i* ∧ *i* < *length F1* ⟶ (*mutex-single* (*F1* ! *i*) *F2*))
proof (*induct F1 arbitrary*: *F2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a F1*)
then show *?case*

using *less-Suc-eq-0-disj* **by** *auto*
qed

4.3.3 Add to lemmas

lemma *length-add-to-now-single*:
shows *length* (*add-to-now-single f L*) = (*if L*=[] *then 0 else 2*∗(*length L*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *add-to-now-single-def* **by** *simp*
next
case (*Cons a L*)
then show *?case* **unfolding** *add-to-now-single-def* **by** *simp*
qed

lemma *length-add-to-next-single*:
shows *length* (*add-to-next-single f L*) = (*if L*=[] *then 0 else 2*∗(*length L*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *add-to-next-single-def* **by** *simp*
next
case (*Cons a L*)
then show *?case* **unfolding** *add-to-next-single-def* **by** *simp*
qed

lemma *length-add-to-next-single-a*:
shows *length* (*add-to-next-single-a f L*) = (*if L*=[] *then 0 else 2*∗(*length L*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *add-to-next-single-a-def* **by** *simp*
next
case (*Cons a L*)
then show *?case* **unfolding** *add-to-next-single-a-def* **by** *simp*
qed

lemma *length-add-to-now-next-gen*:
assumes *length xs* = *length ys*
length L1 = *length L2*
shows *length* (*foldr add-to-now-single xs L1*) =
length (*foldr add-to-next-single ys L2*)
using *assms*
proof (*induct xs ys rule*: *list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
using *Cons* **unfolding** *add-to-now-single-def add-to-next-single-def*

by *simp*
qed

lemma *length-add-to-now-next-a-gen*:
assumes *length xs = length ys*
length L1 = length L2
shows *length* (*foldr add-to-now-single xs L1*) =
length (*foldr add-to-next-single-a ys L2*)
using *assms*
proof (*induct xs ys rule*: *list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
using *Cons* **unfolding** *add-to-now-single-def add-to-next-single-a-def*
by *simp*
qed

lemma *length-add-to-now-next*:
assumes *length xs = length ys*
shows *length* (*foldr add-to-now-single xs* [*itrue* ,*ifalse*]) =
length (*foldr add-to-next-single ys* [*ifalse*,*ifalse*])
using *assms*
by (*simp add*: *length-add-to-now-next-gen*)

lemma *length-add-to-now-next-a*:
assumes *length xs = length ys*
shows *length* (*foldr add-to-now-single xs* [*itrue* ,*ifalse*]) =
length (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])
using *assms*
by (*simp add*: *length-add-to-now-next-a-gen*)

lemma *add-to-now-single-to-big-ior-inv*:
assumes ⊢ *big-ior L*
shows ⊢ *big-ior* (*add-to-now-single f L*)
unfolding *add-to-now-single-def* **using** *assms*
big-ior-append[*of* (*map* (λ*x*. (*f iand x*)) *L*) (*map* (λ*x*. (*inot f iand x*)) *L*)]
by (*auto simp add*: *big-ior-defs*)
(*metis iand-defs nth-map*)

lemma *big-ior-foldr-add-to-now-single-inv-a*:
shows ⊢ *big-ior* (*foldr add-to-now-single xs* [*itrue*])
proof (*induct xs*)
case *Nil*
then show *?case*
using *big-ior-defs* **by** *fastforce*
next

case (*Cons a xs*)
then show *?case*
  using *Cons add-to-now-single-to-big-ior-inv* **by** *auto*
qed

lemma *big-ior-foldr-add-to-now-single-inv*:
shows ⊢ *big-ior* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
proof (*induct xs*)
case *Nil*
then show *?case*
using *big-ior-defs* **by** *fastforce*
next
case (*Cons a xs*)
then show *?case*
  using *Cons add-to-now-single-to-big-ior-inv* **by** *auto*
qed

lemma *add-to-now-single-to-mutex-inv*:
 assumes *mutex L*
 shows *mutex* (*add-to-now-single f L*)
proof −
 have *1*: *mutex* (*map* (λ*x*. *f iand x*) *L*)
  using *assms* **unfolding** *mutex-def* **by** *auto*
 have *2*: *mutex* (*map* (λ*x*. *inot f iand x*) *L*)
  using *assms* **unfolding** *mutex-def* **by** *auto*
 have *3*: *foldr* (λ*x*. (∧) (*mutex-single x* (*map* (λ*x*. (*inot f*) *iand x*) *L*))) (*map* (λ*x*. *f iand x*) *L*) *True*
  ⟷ (∀ *i*. *0* ≤ *i* ∧ *i* < *length* (*map* (λ*x*. *f iand x*) *L*) ⟶
  mutex-single (*map* (λ*x*. *f iand x*) *L* ! *i*) (*map* (λ*x*. (*inot f*) *iand x*) *L*))
  using *unfold-mutex-single*[*of* (*map* (λ*x*. (*inot f*) *iand x*) *L*) (*map* (λ*x*. *f iand x*) *L*)]
  by *blast*
 have *4*: (∀ *i*. *0* ≤ *i* ∧ *i* < *length* (*map* (λ*x*. *f iand x*) *L*) ⟶
  mutex-single (*map* (λ*x*. *f iand x*) *L* ! *i*) (*map* (λ*x*. (*inot f*) *iand x*) *L*))
  using *assms* **unfolding** *mutex-single-def* **by** *auto*
 show *?thesis* **unfolding** *add-to-now-single-def* **using** *assms*
  mutex-append[*of* (*map* (λ*x*. *f iand x*) *L*) (*map* (λ*x*. (*inot f*) *iand x*) *L*)]
 using *1 2 3 4* **by** *blast*
qed

lemma *mutex-foldr-add-to-now-single-inv-a*:
shows *mutex* (*foldr add-to-now-single xs* [*itrue*])
proof (*induct xs*)
case *Nil*
then show *?case* **unfolding** *mutex-def* **by** *simp*
next
case (*Cons a xs*)
then show *?case* **by** (*simp add*: *Cons.hyps add-to-now-single-to-mutex-inv*)
qed

lemma *mutex-foldr-add-to-now-single-inv*:

shows *mutex* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
proof (*induct xs*)
case *Nil*
then show *?case* **unfolding** *mutex-def* **by** *simp*
next
case (*Cons a xs*)
then show *?case* **by** (*simp add*: *Cons.hyps add-to-now-single-to-mutex-inv*)
qed

lemma *add-to-now-single-to-state-pitl-list*:
assumes *state-pitl f*
state-pitl-list L
shows *state-pitl-list* (*add-to-now-single f L*)
using *assms* **unfolding** *add-to-now-single-def*
by (*simp add*: *inot-d-def state-pitl-list-append state-pitl-list-map*)

lemma *state-pitl-foldr-add-to-now-single-inv-a*:
assumes *state-pitl-list xs*
shows *state-pitl-list* (*foldr add-to-now-single xs* [*itrue*])
using *assms*
proof (*induct xs*)
case *Nil*
then show *?case* **using** *Nil*
by (*simp add*: *inot-d-def itrue-d-def state-pitl-list-def*)
next
case (*Cons a xs*)
then show *?case*
by *simp*
(*metis add-to-now-single-to-state-pitl-list append-Cons append-Nil length-Cons nth-Cons-0*
state-pitl-list-append state-pitl-list-defs zero-less-Suc)
qed

lemma *state-pitl-foldr-add-to-now-single-inv*:
assumes *state-pitl-list xs*
shows *state-pitl-list* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *assms*
proof (*induct xs*)
case *Nil*
then show *?case* **using** *Nil*
by (*simp add*: *inot-d-def itrue-d-def state-pitl-list-def*)
next
case (*Cons a xs*)
then show *?case*
by *simp*
(*metis add-to-now-single-to-state-pitl-list append-Cons append-Nil length-Cons nth-Cons-0*
state-pitl-list-append state-pitl-list-defs zero-less-Suc)
qed

```
lemma det-big-ior-ieqv-nondet-big-ior-a:
assumes length L1 = length L2
shows ⊢ big-ior ( map2 (λ x y. x iand (next  y))
                  (foldr (add-to-now-single) L1 [itrue])
                  (foldr (add-to-next-single) L2 [ifalse]))
        ieqv
        big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
   proof (cases xs = [] ∧ ys = [] )
   case True
   then show ?thesis using Cons
    unfolding add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def
     by simp
   next
   case False
   then show ?thesis
     proof −
      have 1: (foldr add-to-now-single (x # xs) [itrue]) =
              ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
         by (simp add: add-to-now-single-def)
      have 2: (foldr add-to-next-single (y # ys) [ifalse]) =
              ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse]))@
                                (foldr add-to-next-single ys [ifalse]) )
        by (simp add: add-to-next-single-def)
      have 3: length xs = length ys
         by (simp add: Cons.hyps(1))
      have 4: ⊢ big-ior (map2 (λx y. x iand next  y)
                        (foldr add-to-now-single xs [itrue])
                        (foldr add-to-next-single ys [ifalse])) ieqv
                 big-ior (map2 (λx y. x iand next y) xs ys)
        using 3 Cons.hyps(2) False by fastforce
      have 5: big-ior (map2 (λx y. x iand next  y)
                      (foldr add-to-now-single (x # xs) [itrue])
                      (foldr add-to-next-single (y # ys) [ifalse])) =
              big-ior (map2 (λx y. x iand next  y)
                      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
                      ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse]))@
                                        (foldr add-to-next-single ys [ifalse]) )
                       )
        using 1 2 by presburger
      have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) =
```

length (*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*]))
by (*simp add*: *3 length-add-to-now-next-gen*)
have *7*: *length* (*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*])) =
length (*foldr add-to-next-single ys* [*ifalse*])
using *6* **by** *auto*
have *75*: *length* (*foldr add-to-now-single xs* [*itrue*]) =
length (*foldr add-to-next-single ys* [*ifalse*])
using *7* **by** *force*
have *8*: ⊢ *big-ior* (*map2* (λ*x y*. *x iand next y*)
((*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*])) @
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*])))
((*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*]))@
(*foldr add-to-next-single ys* [*ifalse*]))
) *ieqv*
big-ior ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*]))
)
using *6* **by** *force*
have *9*: ⊢ *big-ior* ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*]))
) *ieqv*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*])))
using *big-ior-append* **by** *blast*
have *10*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* (*y ior y1*))
((*foldr add-to-now-single xs* [*itrue*]))
((*foldr add-to-next-single ys* [*ifalse*]))))
using *map2-map-a*[*of* (*foldr add-to-now-single xs* [*itrue*])
(*foldr add-to-next-single ys* [*ifalse*]) *x y*] **using** *7* **by** *auto*
have *11*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*]))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. ((*inot x*) *iand x1*) *iand next y1*)

```
                    ( (foldr add-to-now-single xs [itrue]))
                      (foldr add-to-next-single ys [ifalse]) ))
        using map2-map-b[of (foldr add-to-now-single xs [itrue])
          (foldr add-to-next-single ys [ifalse]) x] 75
       by auto
     have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue])
       using big-ior-foldr-add-to-now-single-inv-a by metis
     have 13: 0 < length (foldr add-to-now-single xs [itrue])
       using 12 big-ior-empty by auto
     have 14: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single ys [ifalse])))) ior
              (big-ior (map2 (λx y. x iand next y)
                    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                                  (foldr add-to-next-single ys [ifalse]) ))
              ieqv
              (x iand next y) ior
              (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single ys [ifalse]))))
        using big-ior-split-off[of ( (foldr add-to-now-single xs [itrue]))
          ( (foldr add-to-next-single ys [ifalse])) x y] 12 13
       by (simp add: 75 map2-map-b)
     have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))  ieqv
              (x iand next y) ior
               big-ior (map2 (λx y. x iand next y) xs  ys)
        using big-ior-Cons-alt by auto
    have 16: ⊢ (x iand next y) ior
              (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single ys [ifalse])))) ieqv
            big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
       using 4 15 by auto
     show ?thesis
     using 10 14 16 5 6 9 by force
  qed
  qed
qed

lemma det-big-ior-ieqv-nondet-big-ior-a-a:
assumes length L1 = length L2
shows ⊢ big-ior ( map2 (λ x y. x iand (next  y))
              (foldr (add-to-now-single) L1 [itrue])
              (foldr (add-to-next-single-a) L2 [ifalse]))
       ieqv
       big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
```

```
next
case (Cons x xs y ys)
then show ?case
   proof (cases xs = [] ∧ ys = [] )
   case True
   then show ?thesis using Cons
    unfolding add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def
     by simp
   next
   case False
   then show ?thesis
     proof −
      have 1: (foldr add-to-now-single (x # xs) [itrue]) =
              ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
         by (simp add: add-to-now-single-def)
      have 2: (foldr add-to-next-single-a (y # ys) [ifalse]) =
              ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
                                (foldr add-to-next-single-a ys [ifalse]) )
        by (simp add: add-to-next-single-a-def)
      have 3: length xs = length ys
         by (simp add: Cons.hyps(1))
      have 4: ⊢ big-ior (map2 (λx y. x iand next  y)
                         (foldr add-to-now-single xs [itrue])
                         (foldr add-to-next-single-a ys [ifalse])) ieqv
               big-ior (map2 (λx y. x iand next y) xs ys)
        using 3 Cons.hyps(2) False by fastforce
      have 5: big-ior (map2 (λx y. x iand next  y)
                      (foldr add-to-now-single (x # xs) [itrue])
                      (foldr add-to-next-single-a (y # ys) [ifalse])) =
             big-ior (map2 (λx y. x iand next  y)
                      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
                    ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
                                        (foldr add-to-next-single-a ys [ifalse]) )
                     )
        using 1 2 by presburger
      have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) =
              length (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
        by (simp add: 3 length-add-to-now-next-a-gen)
      have 7:  length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) =
               length (foldr add-to-next-single-a ys [ifalse])
         using 6 by auto
      have 75: length (foldr add-to-now-single xs [itrue]) =
               length (foldr add-to-next-single-a ys [ifalse])
       using 7 by force
      have 8: ⊢ big-ior (map2 (λx y. x iand next y)
                      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
                    ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
```

```
                                        (foldr add-to-next-single-a ys [ifalse]) )
                        ) ieqv
                big-ior ( (map2 (λx y. x iand next y)
                            (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                          (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
                          @
                            (map2 (λx y. x iand next y)
                            (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                            (foldr add-to-next-single-a ys [ifalse]) )
                          )
            using 6 by force
        have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
                            (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                          (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
                          @
                            (map2 (λx y. x iand next y)
                            (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                            (foldr add-to-next-single-a ys [ifalse]) )
                        ) ieqv
                (big-ior  (map2 (λx y. x iand next y)
                            (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                          (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)) ior
                (big-ior (map2 (λx y. x iand next y)
                            (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                                        (foldr add-to-next-single-a ys [ifalse]) ))
          using big-ior-append by blast
        have 10: ⊢ (big-ior  (map2 (λx y. x iand next y)
                            (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                          (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)) ieqv
                    (big-ior  (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
                            ( (foldr add-to-now-single xs [itrue]))
                            ( (foldr add-to-next-single-a ys [ifalse]))))
              using map2-map-a-a[of (foldr add-to-now-single xs [itrue])
            (foldr add-to-next-single-a ys [ifalse]) x y ] using 7 by auto
        have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
                            (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                                        (foldr add-to-next-single-a ys [ifalse]) )) ieqv
                (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
                            ( (foldr add-to-now-single xs [itrue]))
                              (foldr add-to-next-single-a ys [ifalse]) ))
            using map2-map-b[of (foldr add-to-now-single xs [itrue])
              (foldr add-to-next-single-a ys [ifalse]) x] 75
          by auto
        have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue])
          using big-ior-foldr-add-to-now-single-inv-a by metis
        have 13: 0 < length (foldr add-to-now-single xs [itrue])
```

```
      using 12 big-ior-empty by auto
    have 14: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single-a ys [ifalse])))) ior
             (big-ior (map2 (λx y. x iand next y)
                    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                                  (foldr add-to-next-single-a ys [ifalse]) ))
             ieqv
             (x iand next y) ior
             (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single-a ys [ifalse]))))
       using big-ior-split-off-a[of ( (foldr add-to-now-single xs [itrue]))
         ( (foldr add-to-next-single-a ys [ifalse])) x y] 12 13
      by (simp add: 75 map2-map-b)
    have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))  ieqv
             (x iand next y) ior
              big-ior (map2 (λx y. x iand next y) xs  ys)
       using big-ior-Cons-alt by auto
   have 16: ⊢ (x iand next y) ior
             (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single-a ys [ifalse])))) ieqv
           big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
      using 4 15 by auto
    show ?thesis
    using 10 14 16 5 6 9 by force
  qed
  qed
qed

lemma det-big-ior-ieqv-nondet-big-ior:
assumes length L1 = length L2
shows ⊢ big-ior ( map2 (λ x y. x iand (next  y))
             (foldr (add-to-now-single) L1 [itrue,ifalse])
             (foldr (add-to-next-single) L2 [ifalse,ifalse]))
      ieqv
      big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis using Cons
   unfolding add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def
    by simp
```

```
next
case False
then show ?thesis
 proof −
  have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
         ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
           (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
     by (simp add: add-to-now-single-def)
  have 2: (foldr add-to-next-single (y # ys) [ifalse,ifalse]) =
         ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
                          (foldr add-to-next-single ys [ifalse,ifalse]) )
    by (simp add: add-to-next-single-def)
  have 3: length xs = length ys
     by (simp add: Cons.hyps(1))
  have 4: ⊢ big-ior (map2 (λx y. x iand next  y)
                   (foldr add-to-now-single xs [itrue,ifalse])
                   (foldr add-to-next-single ys [ifalse,ifalse])) ieqv
           big-ior (map2 (λx y. x iand next y) xs ys)
    using 3 Cons.hyps(2) False by fastforce
  have 5: big-ior (map2 (λx y. x iand next  y)
                 (foldr add-to-now-single (x # xs) [itrue,ifalse])
                 (foldr add-to-next-single (y # ys) [ifalse,ifalse])) =
          big-ior (map2 (λx y. x iand next  y)
                 ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
                   (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
                 ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
                                  (foldr add-to-next-single ys [ifalse,ifalse]) )
                  )
    using 1 2 by presburger
  have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
         length (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))
    by (simp add: 3 length-add-to-now-next)
  have 7:  length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
          length (foldr add-to-next-single ys [ifalse,ifalse])
     using 6 by auto
  have 75: length (foldr add-to-now-single xs [itrue, ifalse]) =
          length (foldr add-to-next-single ys [ifalse, ifalse])
   using 7 by force
  have 8: ⊢ big-ior (map2 (λx y. x iand next y)
                 ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
                   (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
                 ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
                                  (foldr add-to-next-single ys [ifalse,ifalse]) )
                  ) ieqv
            big-ior ( (map2 (λx y. x iand next y)
                      (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                      (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
                    @
                      (map2 (λx y. x iand next y)
                      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
```

(*foldr add-to-next-single ys* [*ifalse*,*ifalse*]))
)
using *6* **by** *force*
have *9*: ⊢ *big-ior* ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*,*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single ys* [*ifalse*,*ifalse*]))
) *ieqv*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*,*ifalse*])))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single ys* [*ifalse*,*ifalse*])))
using *big-ior-append* **by** *blast*
have *10*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*,*ifalse*]))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* (*y ior y1*))
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single ys* [*ifalse*,*ifalse*]))))
using *map2-map-a*[*of* (*foldr add-to-now-single xs* [*itrue*,*ifalse*])
(*foldr add-to-next-single ys* [*ifalse*,*ifalse*]) *x y*] **using** *7* **by** *auto*
have *11*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single ys* [*ifalse*,*ifalse*]))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. ((*inot x*) *iand x1*) *iand next y1*)
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single ys* [*ifalse*,*ifalse*])))
using *map2-map-b*[*of* (*foldr add-to-now-single xs* [*itrue*,*ifalse*])
(*foldr add-to-next-single ys* [*ifalse*,*ifalse*]) *x*] *75*
by *auto*
have *12*: ⊢ *big-ior* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *big-ior-foldr-add-to-now-single-inv* **by** *metis*
have *13*: *0* < *length* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *12 big-ior-empty* **by** *auto*
have *14*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* (*y ior y1*))
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single ys* [*ifalse*,*ifalse*])))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single ys* [*ifalse*,*ifalse*])))
ieqv
(*x iand next y*) *ior*
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next y1*)
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single ys* [*ifalse*,*ifalse*]))))

```
        using big-ior-split-off[of ( (foldr add-to-now-single xs [itrue,ifalse]))
          ( (foldr add-to-next-single ys [ifalse,ifalse])) x y] 12 13
        by (simp add: 75 map2-map-b)
      have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))  ieqv
                (x iand next y) ior
                 big-ior (map2 (λx y. x iand next y) xs  ys)
        using big-ior-Cons-alt by auto
     have 16: ⊢ (x iand next y) ior
                (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                        ( (foldr add-to-now-single xs [itrue,ifalse]))
                        ( (foldr add-to-next-single ys [ifalse,ifalse]))))  ieqv
              big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
       using 4 15 by auto
      show ?thesis
      using 10 14 16 5 6 9 by force
   qed
   qed
qed

lemma det-big-ior-ieqv-nondet-big-ior-b:
assumes length L1 = length L2
shows ⊢ big-ior ( map2 (λ x y. x iand (next  y))
                (foldr (add-to-now-single) L1 [itrue,ifalse])
                (foldr (add-to-next-single-a) L2 [ifalse,ifalse]))
        ieqv
        big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
   proof (cases xs = [] ∧ ys = [] )
   case True
   then show ?thesis using Cons
    unfolding add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def
     by simp
   next
   case False
   then show ?thesis
     proof −
      have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
              ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
                (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
         by (simp add: add-to-now-single-def)
      have 2: (foldr add-to-next-single-a (y # ys) [ifalse,ifalse]) =
              ( (map (λx1. (if y =ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse,ifalse]))@
                                (foldr add-to-next-single-a ys [ifalse,ifalse]) )
        by (simp add: add-to-next-single-a-def)
```

have *3*: *length xs = length ys*
by (*simp add*: *Cons.hyps*(*1*))
have *4*: ⊢ *big-ior* (*map2* (λ*x y*. *x iand next y*)
(*foldr add-to-now-single xs* [*itrue*,*ifalse*])
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])) *ieqv*
big-ior (*map2* (λ*x y*. *x iand next y*) *xs ys*)
using *3 Cons.hyps*(*2*) *False* **by** *fastforce*
have *5*: *big-ior* (*map2* (λ*x y*. *x iand next y*)
(*foldr add-to-now-single* (*x # xs*) [*itrue*,*ifalse*])
(*foldr add-to-next-single-a* (*y # ys*) [*ifalse*,*ifalse*])) =
big-ior (*map2* (λ*x y*. *x iand next y*)
((*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) @
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])))
((*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))@
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
)
using *1 2* **by** *presburger*
have *6*: *length* (*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) =
length (*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
by (*simp add*: *3 length-add-to-now-next-a*)
have *7*: *length* (*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) =
length (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])
using *6* **by** *auto*
have *75*: *length* (*foldr add-to-now-single xs* [*itrue*, *ifalse*]) =
length (*foldr add-to-next-single-a ys* [*ifalse*, *ifalse*])
using *7* **by** *force*
have *8*: ⊢ *big-ior* (*map2* (λ*x y*. *x iand next y*)
((*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) @
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])))
((*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))@
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
) *ieqv*
big-ior ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
)
using *6* **by** *force*
have *9*: ⊢ *big-ior* ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)

(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
) *ieqv*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y* = *ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
using *big-ior-append* **by** *blast*
have *10*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y* = *ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* ((*if y* = *ifalse then y1 else y ior y1*)))
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))))
using *map2-map-a-a*[*of* (*foldr add-to-now-single xs* [*itrue*,*ifalse*])
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]) *x y*] **using** *7* **by** *auto*
have *11*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. ((*inot x*) *iand x1*) *iand next y1*)
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
using *map2-map-b*[*of* (*foldr add-to-now-single xs* [*itrue*,*ifalse*])
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]) *x*] *75*
by *auto*
have *12*: ⊢ *big-ior* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *big-ior-foldr-add-to-now-single-inv* **by** *metis*
have *13*: *0* < *length* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *12 big-ior-empty* **by** *auto*
have *14*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* (*if y*= *ifalse then y1 else y ior y1*))
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
ieqv
(*x iand next y*) *ior*
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next y1*)
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))))
using *big-ior-split-off-a*[*of* ((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])) *x y*] *12 13*
by (*simp add*: *75 map2-map-b*)
have *15*: ⊢ *big-ior* (*map2* (λ*x y*. *x iand next y*) (*x* # *xs*) (*y* # *ys*)) *ieqv*
(*x iand next y*) *ior*
big-ior (*map2* (λ*x y*. *x iand next y*) *xs ys*)

```
          using big-ior-Cons-alt by auto
      have 16: ⊢ (x iand next y) ior
                  (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                          ( (foldr add-to-now-single xs [itrue,ifalse]))
                          ( (foldr add-to-next-single-a ys [ifalse,ifalse])))) ieqv
                 big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
         using 4 15 by auto
       show ?thesis
       using 10 14 16 5 6 9 by force
    qed
    qed
qed


lemma mutex-single-filter-sat:
 assumes mutex-single f L1
 shows mutex-single f ( filter (λx. (∃ σ. σ ⊨ x)) L1)
using assms
proof (induct L1)
case Nil
then show ?case by simp
next
case (Cons a L1)
then show ?case
by (simp add: mutex-single-Cons)
qed

lemma mutex-filter-sat:
 assumes mutex L1
 shows mutex ( filter (λx. (∃ σ. σ ⊨ x)) L1)
using assms
proof (induct L1)
case Nil
then show ?case unfolding mutex-def by simp
next
case (Cons a L1)
then show ?case
by (simp add: mutex-Cons mutex-single-filter-sat)
qed


lemma state-pitl-list-filter-sat:
 assumes state-pitl-list L
 shows state-pitl-list ( filter (λx. (∃ σ. σ ⊨ x)) L)
using assms
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
```

then show *?case*
by (*simp add*: *state-pitl-list-def*)
qed

lemma *big-ior-itrue*:
$\vdash$ *big-ior* [*itrue*]
by (*simp add*: *big-ior-defs*)

lemma *mutex-itrue*:
mutex [*itrue*]
by (*simp add*: *mutex-Cons mutex-empty mutex-single-empty*)

lemma *big-ior-mutex-start*:
($\vdash$ *big-ior* [*ifalse*,*itrue*]) $\wedge$ *mutex* [*ifalse*,*itrue*]
unfolding *big-ior-def mutex-def*
using *less-Suc-eq* **by** *force*

4.3.4 GNF lemmas

lemma *GNF-defs*:
assumes $\vdash$ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y. x iand* (*next y*)) *F-now F-next*)))
state-pitl F-e
state-pitl-list F-now
length F-now = *length F-next*
shows ($\sigma \models F$) $\longleftrightarrow$
((*nlength* σ = *0* $\wedge$ ((*NNil* (*nfirst* σ)) $\models$ *F-e*)) $\vee$
(*nlength* σ > *0* $\wedge$
($\exists$ *i*<*length F-now*. ((*NNil* (*nfirst* σ)) $\models$ *F-now* !*i*) $\wedge$ ((*ndropn 1* σ) $\models$ *F-next* !*i*))))
proof −
have *1*: ($\sigma \models F$) $\longleftrightarrow$
($\sigma \models$ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y. x iand* (*next y*)) *F-now F-next*))))

using *assms itl-ieq* **by** *blast*
have *2*: ($\sigma \models$ (*F-e iand empty*)) $\longleftrightarrow$
(*nlength* σ = *0* $\wedge$ ((*NNil* (*nfirst* σ)) $\models$ *F-e*))
using *assms state-pitl-defs empty-defs iand-defs* **by** *blast*
have *3*: *length* (*map2* (λ*x y. x iand next y*) *F-now F-next*) = *length F-now*
by (*simp add*: *assms*(*4*))
have *4*: ($\sigma \models$ (*big-ior* (*map2* (λ *x y. x iand* (*next y*)) *F-now F-next*))) $\longleftrightarrow$
($\exists$ *i*<*length F-now*. ($\sigma \models$ (λ *x y. x iand* (*next y*)) (*F-now*!*i*) (*F-next*!*i*)))
by (*simp add*: *assms*(*4*) *big-ior-map2-defs*)
have *5*: ($\exists$ *i*<*length F-now*. ($\sigma \models$ (λ *x y. x iand* (*next y*)) (*F-now*!*i*) (*F-next*!*i*))) $\longleftrightarrow$
(*nlength* σ > *0* $\wedge$
($\exists$ *i*<*length F-now*. ((*NNil* (*nfirst* σ)) $\models$ *F-now* !*i*) $\wedge$ ((*ndropn 1* σ) $\models$ *F-next* !*i*)))

by *simp*
(*metis assms*(*3*) *co.enat.exhaust-sel enat-le-plus-same*(*2*) *not-one-le-zero plus-1-eSuc*(*2*)
state-pitl-defs state-pitl-list-defs)
show *?thesis*
using *1 2 4 5* **by** *auto*
qed

lemma *big-ior-iand-next-big-iand-iimp*:
assumes (⊢ *big-ior F*)
mutex F
length F = *length G*
shows ⊢ *big-ior* (*map2* (λ*x y*. *x iand* (*next y*)) *F G*) *ieqv*
big-iand (*map2* (λ*x y*. *x iimp* (*next y*)) *F G*)
using *assms* **unfolding** *valid-def mutex-def*
apply (*auto simp add*: *big-ior-defs big-iand-defs map2-iand-next-defs*)
apply (*metis antisym-conv3*)
by *meson*

lemma *inot-big-iand-next-iimp-big-iand-iimp-inot*:
assumes (⊢ *big-ior F*)
mutex F
length F = *length G*
shows ⊢ *more iimp* (*inot big-iand* (*map2* (λ*x y*. *x iimp* (*next y*)) *F G*) *ieqv*
big-iand (*map2* (λ *x y*. *x iimp* (*next* (*inot y*))) *F G*))
using *assms* **unfolding** *mutex-def*
apply (*auto simp add*: *big-iand-defs big-ior-defs more-d-def*)
apply (*metis linorder-less-linear*)
by *meson*

lemma *inot-big-ior-iand-big-ior-iand-inot*:
assumes (⊢ *big-ior F*)
mutex F
length F = *length G*
shows ⊢ (*inot* (*big-ior* (*map2* (λ*x y*. *x iand* (*y*)) *F G*)) *ieqv*
big-ior (*map2* (λ *x y*. *x iand* ((*inot y*))) *F G*))
using *assms* **unfolding** *mutex-def*
apply (*auto simp add*: *big-ior-defs big-iand-defs map2-iand-inot-defs*)
apply *meson*
by (*metis not-less-iff-gr-or-eq*)

lemma *inot-big-ior-iand-big-ior-iand-next-inot*:
assumes (⊢ *big-ior F*)
mutex F
length F = *length G*
shows ⊢ *more iimp* (*inot* (*big-ior* (*map2* (λ*x y*. *x iand* (*next y*)) *F G*)) *ieqv*

big-ior (*map2* (λ *x y*. *x iand* (*next* (*inot y*))) *F G*))
using *assms* **unfolding** *mutex-def*
apply (*auto simp add*: *big-ior-defs big-iand-defs map2-iand-inot-next-defs more-d-def*)
apply (*metis One-nat-def gr-zeroI map2-iand-inot-next-defs not-one-le-zero semantics-pitl.simps*(*4*))
by (*metis linorder-less-linear*)

lemma *more-iimp-inot-big-ior-big-ior-map-inot*:
assumes (⊢ *big-ior F*)
mutex F
length F = *length G*
shows ⊢ *more iimp*
((*inot* (*big-ior* (*map2* (λ*x y*. *x iand* (*next y*)) *F G*))) *ieqv*
big-ior (*map2* (λ*x y*. *x iand* (*next y*)) *F* (*map* (λ*x*. (*inot x*)) *G*)))
proof −
have *1*: (*map2* (λ*x y*. *x iand* (*next y*)) *F* (*map* (λ*x*. (*inot x*)) *G*)) =
(*map2* (λ*x y*. *x iand* (*next* (*inot y*))) *F G*)
using *map2-map*[*of F G* (λ*x y*. *x iand* (*next y*)) *id* (λ*x*. (*inot x*))]
assms **by** *auto*
show *?thesis*
using *1 assms inot-big-ior-iand-big-ior-iand-next-inot* **by** *auto*
qed

lemma *GNF-iimp-step*:
assumes ⊢ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-pitl F-e
state-pitl-list F-now
length F-now = *length F-next*
⊢ *G ieqv* ((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*)))
state-pitl G-e
state-pitl-list G-now
length G-now = *length G-next*
shows (*σ* ⊨ *F iimp G*) ⟷
(*σ* ⊨ (((*inot F-e*) *ior G-e*) *iand empty*) *ior*
(*big-ior*
(*map2* (λ *x y*. *x iand* (*next y*))
((*add-to-now F-now*) @*G-now*)
((*map* (λ*x*. (*inot x*)) (*add-to-next F-next*)) @ *G-next*))))
proof (*cases nlength σ* = *0*)
case *True*
then show *?thesis*
proof −
have *1*: (*σ* ⊨ *F iimp G*) ⟷ (¬(*σ* ⊨ *F*) ∨ (*σ* ⊨*G*))
by *auto*
have *2*: (¬(*σ* ⊨ *F*) ∨ (*σ* ⊨*G*)) ⟷
(¬((*NNil* (*nfirst σ*)) ⊨ *F-e*) ∨ ((*NNil* (*nfirst σ*)) ⊨ *G-e*))

```
    using
     assms True
     by (metis (mono-tags, lifting) GNF-defs less-numeral-extra(3))
    have 3: state-pitl (inot F-e ior G-e)
     by (simp add: assms(2) assms(6) inot-d-def ior-d-def)
    have 35: state-pitl-list [ifalse, itrue]
       unfolding state-pitl-list-def itrue-d-def inot-d-def by auto
    have 4: state-pitl-list ((add-to-now F-now ) @ G-now)
       by (simp add: add-to-now-def assms(3) assms(7) state-pitl-foldr-add-to-now-single-inv-a
         state-pitl-list-append)
    have 5: length (add-to-now F-now  @ G-now) =
           length (map inot-d (add-to-next F-next ) @ G-next)
      by (simp add: add-to-next-def add-to-now-def assms(4) assms(8) length-add-to-now-next-gen)
    have 6: (¬((NNil (nfirst σ)) ⊨ F-e) ∨ ((NNil (nfirst σ)) ⊨ G-e)) ⟷
          (σ ⊨ ( ( (inot F-e) ior G-e) iand empty) ior
            (big-ior
             (map2 (λ x y. x iand (next y))
               ((add-to-now F-now )@G-now)
               ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) ) ))
       using GNF-defs[of
          ( ( (inot F-e) ior G-e) iand empty) ior
          (big-ior
           (map2 (λ x y. x iand (next y))
           ((add-to-now F-now )@G-now)
           ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) ))
            ( (inot F-e) ior G-e)
                    ((add-to-now F-now )@G-now)
                    ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) σ ]
       using assms
       using 3 4 5 True by force
     show ?thesis
     using 1 2 6 by blast
   qed
next
case False
then show ?thesis
  proof −
   have 8: (σ ⊨ F iimp G) ⟷ (σ ⊨ (inot F) ior G)
      by auto
   have 9: (σ ⊨ (inot F)) ⟷
         (σ ⊨ (inot (big-ior (map2 (λ x y. x iand (next y))  F-now F-next))))
     using False assms(1) by auto
   have 10: ⊢ (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)) ieqv
            big-ior ( map2 (λ x y. x iand (next  y))
               (foldr (add-to-now-single) F-now [itrue])
               (foldr (add-to-next-single) F-next [ifalse]))
    using assms(4) det-big-ior-ieqv-nondet-big-ior-a by auto
   have 101: (σ ⊨ (inot (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))) ⟷
           (σ ⊨ (inot (big-ior ( map2 (λ x y. x iand (next  y))
               (foldr (add-to-now-single) F-now [itrue])
```

```
                (foldr (add-to-next-single) F-next [ifalse])))))
    using 10 by auto
  have 11: ⊢ big-ior (foldr (add-to-now-single) F-now [itrue])
    using big-ior-foldr-add-to-now-single-inv-a by blast
  have 12: mutex (foldr (add-to-now-single) F-now [itrue])
    using mutex-foldr-add-to-now-single-inv-a by blast
  have 13: length (foldr (add-to-now-single) F-now [itrue]) =
          length (foldr (add-to-next-single) F-next [ifalse])
    by (simp add: assms(4) length-add-to-now-next-gen)
  have 14: ⊢ more iimp
           ( (inot (big-ior ( map2 (λ x y. x iand (next  y))
                (foldr (add-to-now-single) F-now [itrue])
                (foldr (add-to-next-single) F-next [ifalse])))) ieqv
             (big-ior
             (map2 (λ x y. x iand (next y))
                  (foldr (add-to-now-single) F-now [itrue])
                  ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))))))
    using 11 12 13 more-iimp-inot-big-ior-big-ior-map-inot by blast
  have 15: (σ ⊨ (inot (big-ior ( map2 (λ x y. x iand (next  y))
                (foldr (add-to-now-single) F-now [itrue])
                (foldr (add-to-next-single) F-next [ifalse])))) ⟷
          (σ ⊨ (big-ior
             (map2 (λ x y. x iand (next y))
                  (foldr (add-to-now-single) F-now [itrue])
                  ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))))))
     using 14 False by auto
     (metis (no-types, lifting) empty-d-def empty-defs inot-defs,
      metis (no-types, lifting) empty-d-def empty-defs inot-defs)
  have 16: (σ ⊨ G) ⟷
          (σ ⊨ (big-ior (map2 (λ x y. x iand (next y))  G-now G-next)))
   using False assms(5) by auto
  have 17:  (σ ⊨ (inot F) ior G) ⟷
          (σ ⊨ (big-ior  (map2 (λ x y. x iand (next y))
                (foldr (add-to-now-single) F-now [itrue])
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))) ior
               (big-ior (map2 (λ x y. x iand (next y))  G-now G-next)))
               using 101 15 16 9 by auto
  have 18: (σ ⊨ (big-ior  (map2 (λ x y. x iand (next y))
                (foldr (add-to-now-single) F-now [itrue])
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))) ior
               (big-ior (map2 (λ x y. x iand (next y))  G-now G-next))) ⟷
          (σ ⊨ (big-ior  (map2 (λ x y. x iand (next y))
                ((foldr (add-to-now-single) F-now [itrue]) @ G-now)
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])) @ G-next) ) ))
  using 13 big-ior-append by auto
  show ?thesis unfolding add-to-now-def add-to-next-def
  using 17 18 False by auto
 qed
qed
```

lemma *GNF-inot-step*:
assumes $\vdash$ *F ieqv ((F-e iand empty) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))*
state-pitl F-e
state-pitl-list F-now
length F-now = length F-next
shows $(\sigma \models (inot\ F)\ ) \longleftrightarrow$
$(\sigma \models (\ (inot\ F\text{-}e)\ iand\ empty)\ ior$
$(big\text{-}ior$
$(map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))$
$((add\text{-}to\text{-}now\ F\text{-}now\ )\ @[ifalse])$
$((map\ (\lambda x.\ (inot\ x))\ (add\text{-}to\text{-}next\ F\text{-}next\ ))\ @\ [ifalse])\ )))$
proof −
have *1*: $(\sigma \models (inot\ F)\ ) \longleftrightarrow (\sigma \models F\ iimp\ ifalse)$
by *auto*
have *2*: $\vdash$ *ifalse ieqv ifalse iand empty ior big-ior (map2 (λx y. x iand next y) [ifalse] [ifalse])*
by (*auto simp add*: *big-ior-defs*)
have *3*: $(\sigma \models F\ iimp\ ifalse) \longleftrightarrow$
$(\sigma \models (\ (\ (inot\ F\text{-}e)\ ior\ ifalse)\ iand\ empty)\ ior$
$(big\text{-}ior$
$(map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))$
$((add\text{-}to\text{-}now\ F\text{-}now\ )\ @\ [ifalse])$
$((map\ (\lambda x.\ (inot\ x))\ (add\text{-}to\text{-}next\ F\text{-}next\ ))\ @\ [ifalse])\ )))$
using *GNF-iimp-step*[*of F F-e F-now F-next ifalse ifalse* [*ifalse*] [*ifalse*] *σ*]
assms
by (*metis 2 length-Cons less-Suc0 list.size*(*3*) *nth-Cons-0 state-pitl.simps*(*1*)
state-pitl-list-defs)
show *?thesis*
using *3* **by** *auto*
qed

lemma *GNF-inot-step-valid*:
assumes $\vdash$ *F ieqv ((F-e iand empty) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))*
state-pitl F-e
state-pitl-list F-now
length F-now = length F-next
shows $\vdash$ $(inot\ F)$ *ieqv*
$(\ (\ (inot\ F\text{-}e)\ iand\ empty)\ ior$
$(big\text{-}ior$
$(map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))$
$((add\text{-}to\text{-}now\ F\text{-}now\ )\ @[ifalse])$
$((map\ (\lambda x.\ (inot\ x))\ (add\text{-}to\text{-}next\ F\text{-}next\ ))\ @\ [ifalse])\ )))$
using *assms GNF-inot-step*[*of F F-e F-now F-next*]
using *itl-ieq* **by** *blast*

lemma *GNF-inot-inot-step-valid*:

assumes $\vdash$ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-pitl F-e
state-pitl-list F-now
length F-now = *length F-next*
shows $\vdash$ (*inot* (*inot F*)) *ieqv*
((*F-e iand empty*) *ior*
(*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
using *assms* **by** *simp*

lemma *GNF-ior-step*:
assumes $\vdash$ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-pitl F-e
state-pitl-list F-now
length F-now = *length F-next*
$\vdash$ *G ieqv* ((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*)))
state-pitl G-e
state-pitl-list G-now
length G-now = *length G-next*
shows ($\sigma \models$ *F ior G*) $\longleftrightarrow$
($\sigma \models$ ((*F-e ior G-e*) *iand empty*) *ior*
(*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) (*F-now* @ *G-now*) (*F-next* @ *G-next*))))
proof −
have *1*: ($\sigma \models$ *F ior G*) $\longleftrightarrow$
($\sigma \models$ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) *ior*
((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*))))
using *assms* **by** *auto*
have *2*: ($\sigma \models$ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) *ior*
((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*)))) $\longleftrightarrow$
($\sigma \models$ ((*F-e ior G-e*) *iand empty*) *ior*
(*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) (*F-now* @ *G-now*) (*F-next* @ *G-next*))))
using *assms big-ior-map2-append* **by** *fastforce*
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *GNF-schop-step*:
assumes $\vdash$ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-pitl F-e
state-pitl-list F-now
length F-now = *length F-next*
$\vdash$ *G ieqv* ((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*)))
state-pitl G-e
state-pitl-list G-now
length G-now = *length G-next*
shows ($\sigma \models$ *F schop G*) $\longleftrightarrow$
($\sigma \models$ (*F-e iand G-e*) *iand empty ior*
(*big-ior* (*map2* (λ *x y*. *x iand next y*)

```
                    (F-now @ (map (λx. x iand F-e) G-now))
                    ((map (λx. x schop G) F-next) @  G-next))) )
proof (cases nlength σ = 0)
case True
then show ?thesis
 proof −
  have 1: (σ ⊨ F schop G) ⟷ (σ ⊨ F) ∧ (σ ⊨ G)
     using True by (auto simp add: ntaken-all zero-enat-def)
  have 2: (σ ⊨ F) ∧ (σ ⊨ G) ⟷ ((NNil (nfirst σ)) ⊨ F-e) ∧ ((NNil (nfirst σ)) ⊨ G-e)
    using GNF-defs True assms
    by (metis less-numeral-extra(3))
  have 3: state-pitl (F-e iand G-e)
    by (simp add: assms(2) assms(6) iand-d-def inot-d-def ior-d-def)
  have 4: state-pitl-list (F-now @ map (λx. x iand F-e) G-now)
    using assms state-pitl-list-append state-pitl-list-map-alt by blast
  have 5: length (F-now @ map (λx. x iand F-e) G-now) =
         length (map (λx.  x schop G ) F-next @ G-next)
     by (simp add: assms(4) assms(8))
  have 6: ((NNil (nfirst σ)) ⊨ F-e) ∧ ((NNil (nfirst σ)) ⊨ G-e) ⟷
         (σ ⊨ (F-e iand G-e) iand empty ior
            (big-ior (map2 (λ x y. x iand next y)
                    (F-now @ (map (λx. x iand F-e) G-now))
                    ((map (λx. x schop G) F-next) @  G-next))) )
      using GNF-defs[of (F-e iand G-e) iand empty ior
            (big-ior (map2 (λ x y. x iand next y)
                    (F-now @ (map (λx. x iand F-e) G-now))
                    ((map (λx. x schop G) F-next) @  G-next)))
             (F-e iand G-e) (F-now @ (map (λx. x iand F-e) G-now))
             ((map (λx. x schop G) F-next) @  G-next) σ]
    using assms
    by (simp add: 3 4 True)
  show ?thesis
  using 1 2 6 by presburger
 qed
next
case False
 assume a: nlength σ ≠ 0
 show ?thesis (is ?lhs ⟷ ?rhs)
  proof
    assume l:?lhs
    show ?rhs
    proof −
     have 7: (∃ i. enat i ≤ nlength σ ∧ (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G))
      using l by simp
     obtain i where 8: enat i ≤ nlength σ ∧ (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G)
       using 7 by blast
     have 9: i=0 ⟹ ?rhs
       proof −
         assume b:i=0
        have 10: (ntaken 0 σ ⊨ F) ∧ (σ ⊨ G)
```

```
    using 8 b by auto
   have 11: ( σ ⊨ F-e)
    using assms using 8 by  simp
    (metis (mono-tags, lifting) GNF-defs assms(1) b less-numeral-extra(3) min-enat-simps(3)
    ntaken-nfirst ntaken-nlength state-pitl-defs zero-enat-def)
   have 12: σ ⊨ F-e iand big-ior (map2 (λ x y. x iand next y) G-now G-next)
     using 10 11 a assms(5) by force
   have 13: σ ⊨ big-ior (map2 (λ x y. x iand next y)
              (map (λx. x iand F-e) G-now) G-next)
    using iand-big-ior-dist 12 assms(8) itl-ieq by blast
  show ?thesis using assms(4) assms(8) big-ior-map2-append[of F-now map (λx.  x schop G ) F-next
    map (λx. x iand F-e) G-now G-next] by (simp add: 13)
  qed
 have 14: 0<i ∧ enat i ≤ nlength σ ⟹ ?rhs
  proof −
   assume c: 0<i ∧ enat i ≤ nlength σ
  have 15:  (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G)
   by (simp add: 8)
  have 16:  (ntaken i σ ⊨ (big-ior (map2 (λ x y. x iand next y) F-now F-next)))
    using 8 assms(1) c enat-0-iff(1) by auto
  have 17: ((ntaken i σ) ⊨ (big-ior (map2 (λ x y. x iand next y) F-now F-next))) ⟷
         (∃ ia<length F-now. ntaken i σ ⊨ F-now ! ia iand next F-next ! ia)
    using 16 big-ior-map2-defs[of F-now F-next (ntaken i σ) (λ x y. x iand next y)  ]
   assms by blast
  have 18: length F-now = length (map (λx. x schop G) F-next)
   using assms(4) by fastforce
  have 19: (σ ⊨ (big-ior (map2 (λ x y. x iand next y)
              F-now  (map (λx. x schop G) F-next) ))) ⟷
         (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i)
    using 18  big-ior-map2-defs[of F-now (map (λx. x schop G) F-next) σ (λ x y. x iand next y)]
  by blast
  obtain j where  20: j<length F-now ∧ (ntaken i σ ⊨ F-now ! j iand next F-next ! j)
    using 16 17 by blast
  have 21: enat (i−1) ≤ nlength σ − enat (Suc 0)
    by (metis One-nat-def c enat-minus-mono1 idiff-enat-enat)
  have 22: ( (ntaken (i−1) (ndropn (Suc 0) σ) ⊨ F-next ! j))
   by (metis 20 One-nat-def Suc-leI c diff-add iand-defs ntaken-ndropn-swap plus-enat-simps(1)
     semantics-pitl.simps(4))
  have 23: (  (ndropn (i−1) (ndropn (Suc 0) σ) ⊨ G))
   using 8 by (simp add:  c ndropn-ndropn)
  have 24: ( σ ⊨ F-now ! j iand next ((F-next ! j) schop G ) )
    using 20 15 by simp
   (metis 21 22 23 assms(3) ntaken-nfirst state-pitl-defs state-pitl-list-defs)
  have 25: (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i)
     using 20 24 assms(4) by fastforce
  show ?thesis using 18 19 25 assms(8) big-ior-map2-append[of F-now map (λx.  x schop G ) F-next
    map (λx. x iand F-e) G-now G-next ] by simp
 qed
show ?thesis
using 14 8 9 by blast
```

```
  qed
  next
   assume r: ?rhs
   show ?lhs
    proof −
     have 26: σ ⊨ big-ior (map2 (λx y. x iand next y) (F-now @ map (λx. x iand F-e) G-now) (map (λx.
x schop G ) F-next @ G-next))
       using a r by auto
     have 27: length F-now = length (map (λx.  x schop G ) F-next)
       by (simp add: assms(4))
     have 28: length (map (λx. x iand F-e) G-now) = length (G-next)
       using assms(8) by fastforce
     have 29: (σ ⊨ (big-ior (map2 (λx y. x iand next y) (F-now) (map (λx.  x schop G ) F-next))))
                  ∨
              (σ ⊨   ((big-ior (map2 (λx y. x iand next y) (map (λx. x iand F-e) G-now) (G-next)   ))))
       using 26 27 28 big-ior-map2-append itl-ieq ior-defs by blast
     have 30: (σ ⊨ (big-ior (map2 (λx y. x iand next y) (F-now) (map (λx.  x schop G ) F-next)))) ⟷
           (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i)
       using big-ior-map2-defs[of F-now (map (λx.  x schop G ) F-next) σ (λx y. x iand next y)]
        using 27 by fastforce
     have 31: (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i) ⟹ ?lhs
       proof −
        assume d: (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i)
        obtain i where 32: i<length F-now ∧ (σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i)
         using d by blast
        have 33: (σ ⊨ F-now ! i)
          using 32 iand-defs by blast
        have 34: map (λx.  x schop G ) F-next ! i = (F-next !i) schop G
           using 32 assms(4) by auto
        have 35: 1 ≤ nlength σ ∧ (∃ ia. enat ia ≤ nlength σ − enat (Suc 0) ∧ (ntaken ia (ndropn (Suc 0)
σ) ⊨ F-next ! i) ∧ (ndropn ia (ndropn (Suc 0) σ) ⊨ G))
          using 32 34 by simp
        obtain ia where 36: enat ia ≤ nlength σ − enat (Suc 0) ∧ (ntaken ia (ndropn (Suc 0) σ) ⊨ F-next
! i) ∧ (ndropn ia (ndropn (Suc 0) σ) ⊨ G)
         using 35 by blast
        have 37: (ntaken (ia+1) ( σ) ⊨ (big-ior (map2 (λ x y. x iand (next y))  F-now F-next))) ⟷
              (∃ i<length F-now. ntaken (ia+1) ( σ) ⊨ F-now ! i iand next F-next ! i)
         using assms  big-ior-map2-defs[of F-now F-next (ntaken (ia+1) ( σ)) (λ x y. x iand (next y))] by
blast
        have 38: (ntaken (ia+1)  σ ⊨ F-now ! i iand next F-next ! i)
         by simp
          (metis 32 33 35 36 One-nat-def add.commute assms(3) enat-min-eq enat-ord-simps(1)
           le-add1 ntaken-ndropn-swap ntaken-nfirst one-enat-def plus-1-eq-Suc state-pitl-defs
            state-pitl-list-defs)
        have 39:  (ntaken (ia+1)  σ ⊨ F)
          using 32 37 38 assms(1) by auto
        have 40: (ndropn (ia+1)  σ) ⊨ G
          by (metis 36 One-nat-def add.commute ndropn-ndropn)
        have 41: (ia+1) ≤ nlength σ
          by (metis 35 36 One-nat-def enat-min-eq one-enat-def plus-enat-simps(1))
```

```
      show ?thesis
      using 39 40 41 by auto
    qed
   have 42: (σ ⊨   ((big-ior (map2 (λx y. x iand next y) (map (λx. x iand F-e) G-now) (G-next)   ))))
⟷
          (∃ i<length (map (λx. x iand F-e) G-now). σ ⊨ map (λx. x iand F-e) G-now ! i iand next
G-next ! i)
    using big-ior-map2-defs[of (map (λx. x iand F-e) G-now) G-next σ (λx y. x iand next y)]
     using 28 by fastforce
   have 43: (∃ i<length (map (λx. x iand F-e) G-now). σ ⊨ map (λx. x iand F-e) G-now ! i iand next
G-next ! i) ⟹ ?lhs
    proof −
     assume e: (∃ i<length (map (λx. x iand F-e) G-now). σ ⊨ map (λx. x iand F-e) G-now ! i iand
next G-next ! i)
    obtain i where 44: i<length (map (λx. x iand F-e) G-now) ∧ (σ ⊨ map (λx. x iand F-e) G-now !
i iand next G-next ! i)
       using e by blast
     have 45: (ntaken 0 σ) ⊨ ( G-now ! i) iand F-e
      using assms
      by (metis 28 44  iand-defs ntaken-0 nth-map state-pitl-defs state-pitl-list-defs)
     have 46: (ntaken 0 σ) ⊨ F
       using 45 assms(1) by fastforce
     have 47: (ndropn 0 σ) ⊨ G-now ! i iand next G-next ! i
      using 44 by fastforce
     have 48: ((ndropn 0 σ) ⊨ (big-ior (map2 (λ x y. x iand (next y))  G-now G-next))) ⟷
            (∃ i<length G-now. ndropn 0 σ ⊨ G-now ! i iand next G-next ! i)
      using big-ior-map2-defs[of G-now G-next (ndropn 0 σ)  (λ x y. x iand (next y))]
       using assms(8) by fastforce
     have 49: (ndropn 0 σ) ⊨ G
      using 44 47 48 assms(5) by auto
     show ?thesis
     by (metis 46 49 semantics-pitl.simps(5) zero-enat-def zero-le)
    qed
   show ?thesis
   using 29 30 31 42 43 by fastforce
  qed
 qed
qed


lemma GNF-schopstar-step:
 assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))
        state-pitl F-e
        state-pitl-list F-now
        length F-now = length F-next
 shows   (σ ⊨ schopstar F) ⟷
         (σ ⊨ (itrue iand empty) ior
              (big-ior (map2 (λ x y. x iand next y) F-now (map (λx. x schop (schopstar F)) F-next))))
       (is ?lhs ⟷ ?rhs)
proof −
```

have *1*: $(\sigma \models \mathit{schopstar}\ F) \longleftrightarrow$
$(\exists\, l.\ \mathit{nnth}\ l\ 0 = 0 \wedge \mathit{nfinite}\ l \wedge \mathit{nidx}\ l \wedge \mathit{enat}\ (\mathit{nlast}\ l) = \mathit{nlength}\ \sigma \wedge \mathit{nfinite}\ \sigma \wedge$
$(\forall\, i.\ \mathit{enat}\ i < \mathit{nlength}\ l \longrightarrow (\mathit{nsubn}\ \sigma\ (\mathit{nnth}\ l\ i)\ (\mathit{nnth}\ l\ (\mathit{Suc}\ i)) \models F)))$
by *simp*
have *2*: *?lhs* $\Longrightarrow$ *?rhs*
proof −
assume *l*: *?lhs*
obtain *l* **where** *3*: $\mathit{nnth}\ l\ 0 = 0 \wedge \mathit{nfinite}\ l \wedge \mathit{nidx}\ l \wedge \mathit{enat}\ (\mathit{nlast}\ l) = \mathit{nlength}\ \sigma \wedge \mathit{nfinite}\ \sigma \wedge$
$(\forall\, i.\ \mathit{enat}\ i < \mathit{nlength}\ l \longrightarrow (\mathit{nsubn}\ \sigma\ (\mathit{nnth}\ l\ i)\ (\mathit{nnth}\ l\ (\mathit{Suc}\ i)) \models F))$
using *l 1* **by** *blast*
have *4*: $\mathit{nlength}\ l = 0 \Longrightarrow$ *?rhs*
proof −
assume *n*: $\mathit{nlength}\ l = 0$
have *5*: $\mathit{nlength}\ \sigma = 0$
by (*metis 3 n nnth-nlast the-enat-0 zero-enat-def*)
show *?thesis*
by (*simp add*: *5*)
qed
have *6*: $0 < \mathit{nlength}\ l \Longrightarrow$ *?rhs*
proof −
assume *g*: $0 < \mathit{nlength}\ l$
have *7*: $0 < \mathit{nlength}\ \sigma$
by (*metis 3 g gr-zeroI less-numeral-extra*(*3*) *nfinite-conv-nlength-enat nidx-less-last-1*
nnth-nlast the-enat.simps the-enat-0)
have *8*: $(\mathit{nsubn}\ \sigma\ (\mathit{nnth}\ l\ 0)\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0)) \models F)$
using *3 g* **by** (*metis zero-enat-def*)
have *9*: $(\mathit{nsubn}\ \sigma\ (\mathit{nnth}\ l\ 0)\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))) = (\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma)$
using *3* **by** (*simp add*: *nsubn-def1*)
have *10*: $(\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma) \models F$
using *9 8* **by** *auto*
have *11*: $(\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma) \models ((\ F\text{-}e\ \mathit{iand}\ \mathit{empty}\ )\ \mathit{ior}\ (\mathit{big\text{-}ior}\ (\mathit{map2}\ (\lambda\ x\ y.\ x\ \mathit{iand}\ (\mathit{next}$
$y))\ \ F\text{-}\mathit{now}\ F\text{-}\mathit{next})))$
using *10 assms*(*1*) *itl-ieq* **by** *blast*
have *12*: $(\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma) \models (\mathit{big\text{-}ior}\ (\mathit{map2}\ (\lambda\ x\ y.\ x\ \mathit{iand}\ (\mathit{next}\ y))\ \ F\text{-}\mathit{now}\ F\text{-}\mathit{next}))$
by (*metis 11 3 7 9 One-nat-def eSuc-enat empty-defs enat-defs*(*1*) *g iand-defs ileI1*
ior-defs less-numeral-extra(*1*) *linorder-le-cases nidx-gr-first nsubn-nlength-gr-one*
ntaken-all zero-less-iff-neq-zero)
have *13*: $0 < \mathit{nlength}\ (\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma)$
by (*metis 3 7 9 One-nat-def eSuc-enat enat-defs*(*1*) *g ileI1 less-numeral-extra*(*1*)
linorder-le-cases nidx-gr-first nsubn-nlength-gr-one ntaken-all)
have *14*: $(\mathit{nlength}\ (\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma) > 0 \wedge$
$(\exists\, i{<}\mathit{length}\ F\text{-}\mathit{now}.\ ((\mathit{NNil}\ (\mathit{nfirst}\ (\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma))) \models F\text{-}\mathit{now}\ !i\ ) \wedge$
$((\mathit{ndropn}\ 1\ (\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma)) \models F\text{-}\mathit{next}\ !i))\ )$
using *GNF-defs*[*of F F-e F-now F-next* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)] *12 13*
by (*metis 10 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *order-less-irrefl*)
obtain *i* **where** *15*: $i{<}\mathit{length}\ F\text{-}\mathit{now} \wedge ((\mathit{NNil}\ (\mathit{nfirst}\ (\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma))) \models F\text{-}\mathit{now}\ !i\ ) \wedge$
$((\mathit{ndropn}\ 1\ (\mathit{ntaken}\ (\mathit{nnth}\ l\ (\mathit{Suc}\ 0))\ \sigma)) \models F\text{-}\mathit{next}\ !i)$
using *14* **by** *blast*
have *19*: $((\mathit{ndropn}\ 1\ \sigma) \models F\text{-}\mathit{next}\ !i\ \mathit{schop}\ (\mathit{schopstar}\ F)) \longleftrightarrow$
$(\exists\, \mathit{ia}.\ \mathit{enat}\ \mathit{ia} \leq \mathit{nlength}\ \sigma - \mathit{enat}\ (\mathit{Suc}\ 0) \wedge$

(*ntaken ia* (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*) ∧
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶
(*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*))))
by *simp*
have *191*: *enat* ((*nnth l* (*Suc 0*))−*1*) ≤ *nlength σ* − *enat* (*Suc 0*)
by (*metis 3 One-nat-def eSuc-enat enat-minus-mono1 enat-ord-simps*(*1*) *g idiff-enat-enat ileI1 nidx-all-le-nlast zero-enat-def*)
have *192*: *ntaken* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*
by (*metis 15 191 3 7 One-nat-def Suc-leI diff-add eSuc-enat enat-min-eq g ileI1 less-numeral-extra*(*1*) *nidx-gr-first ntaken-ndropn-swap zero-enat-def*)
have *193*: *nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *0* = *0*
using *enat-defs*(*1*) **by** *auto*
have *194*: *nfinite* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
by (*simp add*: *3*)
have *195*: *nidx* (*ndropn 1 l*)
by (*metis 3 g ileI1 nidx-ndropn one-eSuc one-enat-def*)
have *196*: *nnth* (*ndropn 1 l*) *0* = (*nnth l* (*Suc 0*))
by *simp*
have *197*: *nidx* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
using *nidx-shiftm*[*of* (*ndropn 1 l*) (*nnth l* (*Suc 0*))]
by (*metis 195 196 One-nat-def*)
have *198*: *nlast* (*ndropn* (*Suc 0*) *l*) = *nlast l*
by (*metis Suc-ile-eq g nappend-NNil nappend-ntaken-ndropn nlast-NCons ntaken-0 zero-enat-def*)
have *199*: *nlast* (*nmap* (λ*x*. *x* − *nnth l* (*Suc 0*)) (*ndropn* (*Suc 0*) *l*)) =
nlength σ − *nnth l* (*Suc 0*)
using *nlast-nmap*[*of* (*ndropn* (*Suc 0*) *l*) (λ*x*. *x* − (*nnth l* (*Suc 0*)))]
using *194 nfinite-nmap*
by (*metis 198 3 idiff-enat-enat*)
have *200*: *nlength σ* − *nnth l* (*Suc 0*) = *nlength σ* − *enat* (*Suc 0*) − ((*nnth l* (*Suc 0*))−*1*)
by *auto*
(*metis 3 One-nat-def Suc-pred g ileI1 less-numeral-extra*(*1*) *ndropn-ndropn ndropn-nlength nidx-gr-first one-eSuc one-enat-def plus-1-eq-Suc*)
have *201*: *enat* (*nlast* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))) =
nlength σ − *enat* (*Suc 0*) − ((*nnth l* (*Suc 0*))−*1*)
using *199 200* **by** *presburger*
have *202*: *nlength* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) = *nlength l* − (*Suc 0*)
by *force*
have *203*: (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*)) = (*ndropn* (*nnth l* (*Suc 0*)) *σ*)
by (*metis 3 One-nat-def Suc-pred g ileI1 ndropn-ndropn nidx-gr-first one-eSuc one-enat-def plus-1-eq-Suc zero-less-one*)
have *204*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j* =
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
by *simp*
have *2041*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*) =

nnth l ((Suc j) + (Suc 0)) − (nnth l (Suc 0))
by (*metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap plus-1-eq-Suc*)
have *205*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)) < *nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
using *204 197* **unfolding** *nidx-expand* **by** *auto*
(*metis 204 One-nat-def add.commute eSuc-enat ileI1 plus-1-eq-Suc*)
have *206*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*))
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j*)
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*))) =
(*nsubn* (*ndropn* (*nnth l* (*Suc 0*)) *σ*)
(*nnth l* (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))
(*nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*))))
using *203 204 2041* **by** *simp*
have *207*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* (*ndropn* (*nnth l* (*Suc 0*)) *σ*)
(*nnth l* (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))
(*nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))) =
(*nsubn σ* (*nnth l* (*Suc j*)) (*nnth l* (*Suc* (*Suc j*))))
using *nsubn-ndropn*[*of - - σ* (*nnth l* (*Suc 0*))] **by** *simp*
(*metis* (*no-types*, *lifting*) *197 202 204 205 add.commute add.right-neutral bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first order-less-imp-le plus-1-eq-Suc zero-less-diff*)
have *208*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn σ* (*nnth l* (*Suc j*)) (*nnth l* (*Suc* (*Suc j*)))) ⊨ *F*
using *3*
by (*metis eSuc-enat enat-min g ileI1 one-eSuc plus-1-eSuc*(*2*) *zero-enat-def*)
have *209*: (∀ *i*. *enat i* < *nlength* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) ⟶
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*))
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *i*)
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc i*)) ⊨ *F*))
using *202 206 207 208* **by** *presburger*
have *210*: (*enat* ((*nnth l* (*Suc 0*))−*1*) ≤ *nlength σ* − *enat* (*Suc 0*) ∧
(*ntaken* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*) ∧
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *0* = *0* ∧
nfinite (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) ∧
nidx (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) ∧
enat (*nlast* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))) =
nlength σ − *enat* (*Suc 0*) − *enat* ((*nnth l* (*Suc 0*))−*1*) ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) ⟶
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*))
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *i*)
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc i*)) ⊨ *F*))))
using *191 192 193 194 197 201 209 3* **by** *blast*
have *20*: ((*ndropn 1 σ*) ⊨ *F-next* !*i schop* (*schopstar F*))
using *210 19* **by** *blast*
have *21*: *σ* ⊨ *F-now* ! *i*
by (*metis 15 assms*(*3*) *ntaken-nfirst state-pitl-defs state-pitl-list-defs*)
have *22*: *length F-now* = *length* (*map* (λ*x*. *x schop schopstar F*) *F-next*)

by (*simp add*: *assms*(*4*))
have *23*: (*σ* ⊨ *big-ior* (*map2* (*λx y*. *x iand next y*) *F-now* (*map* (*λx*. *x schop schopstar F*) *F-next*)))
⟷
(∃ *i*<*length F-now*. *σ* ⊨ *F-now* ! *i iand next map* (*λx*. *x schop schopstar F*) *F-next* ! *i*)
using *big-ior-map2-defs*[*of F-now* (*map* (*λx*. *x schop* (*schopstar F*)) *F-next*) *σ* (*λ x y*. *x iand next y*)]
22 **by** *auto*
have *24*: (*σ* ⊨ *F-now* ! *i iand next map* (*λx*. *x schop schopstar F*) *F-next* ! *i*)
by (*metis 15 20 21 7 assms*(*4*) *iand-defs ileI1 nth-map one-eSuc semantics-pitl.simps*(*4*))
show *?thesis* **using** *15 23 24 ior-defs* **by** *blast*
qed
show *?thesis*
using *4 6 zero-less-iff-neq-zero* **by** *blast*
qed
have *1000*: *?rhs* ⟹ *?lhs*
proof −
assume *r*: *?rhs*
have *1001*: *nlength σ* = *0* ∨ (*σ* ⊨ *big-ior* (*map2* (*λx y*. *x iand next y*) *F-now* (*map* (*λx*. *x schop schopstar F*) *F-next*)))
using *r* **by** *simp*
have *1002*: *nlength σ* = *0* ⟹ *?lhs*
proof −
assume *n*: *nlength σ* = *0*
have *1003*: *nnth* (*NNil 0*) *0* = *0*
by (*simp add*: *nnth-NNil*)
have *1004*: *nfinite* (*NNil 0*)
by *simp*
have *1005*: *nidx* (*NNil 0*)
using *Suc-ile-eq nidx-expand* **by** *auto*
have *1006*: *enat* (*nlast* (*NNil 0*)) = *nlength σ*
by (*simp add*: *n zero-enat-def*)
have *1007*: *nfinite σ*
using *n*
by (*simp add*: *nfinite-conv-nlength-enat zero-enat-def*)
have *1008*: (∀ *i*. *enat i* < *nlength* (*NNil 0*) ⟶ (*nsubn σ* (*nnth* (*NNil 0*) *i*) (*nnth* (*NNil 0*) (*Suc i*)) ⊨ *F*))
by *auto*
show *?thesis* **apply** *simp*
using *1003 1005 1006 1007 1008* **by** *blast*
qed
have *1009*: (*σ* ⊨ *big-ior* (*map2* (*λx y*. *x iand next y*) *F-now* (*map* (*λx*. *x schop schopstar F*) *F-next*)))
⟹ *?lhs*
proof −
assume *b*: (*σ* ⊨ *big-ior* (*map2* (*λx y*. *x iand next y*) *F-now* (*map* (*λx*. *x schop schopstar F*) *F-next*)))
have *1010*: *length F-now* = *length* (*map* (*λx*. *x schop schopstar F*) *F-next*)
by (*simp add*: *assms*(*4*))
have *1011*: (∃ *i*<*length F-now*. *σ* ⊨ *F-now* ! *i iand next map* (*λx*. *x schop schopstar F*) *F-next* ! *i*)
using *1010 b big-ior-map2-defs*[*of F-now* (*map* (*λx*. *x schop* (*schopstar F*)) *F-next*) *σ* (*λ x y*. *x iand next y*)]
by *blast*

obtain *i* **where** *1012*: *i<length F-now* ∧ (*σ* ⊨ *F-now* ! *i iand next map* (λ*x*. *x schop schopstar F*) *F-next* ! *i*)
using *1011* **by** *blast*
have *1013*: (*σ* ⊨ *F-now* ! *i*)
using *1012* **by** *auto*
have *1014*: ((*ndropn 1 σ*) ⊨ *F-next* !*i schop* (*schopstar F*))
using *1012 assms*(*4*) **by** *fastforce*
have *1015*: ((*ndropn 1 σ*) ⊨ *F-next* !*i schop* (*schopstar F*)) ⟷
(∃ *ia*. *enat ia* ≤ *nlength σ* − *enat* (*Suc 0*) ∧
(*ntaken ia* (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*) ∧
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*))))
by *simp*
obtain *ia* **where** *1016*: *enat ia* ≤ *nlength σ* − *enat* (*Suc 0*) ∧
(*ntaken ia* (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*) ∧
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*)))
using *1014 1015* **by** *blast*
obtain *l* **where** *1017*: *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*))
using *1016* **by** *blast*
have *1018*: *0<nlength σ*
using *1012* **by** *force*
have *10183*: *nlength σ* − *enat* (*Suc 0*) − *enat ia* = *nlength σ* − *Suc ia*
by (*metis One-nat-def ndropn-ndropn ndropn-nlength plus-1-eq-Suc*)
have *10184*: *Suc ia* ≤ *nlength σ*
by (*metis 1016 1018 One-nat-def eSuc-enat enat-min-eq ileI1 one-eSuc one-enat-def plus-1-eSuc*(*2*))
have *10185*: *nlength σ* − *Suc ia* + *Suc ia* = *nlength σ*
by (*metis 10184 add.commute enat.simps*(*3*) *enat-add-sub-same le-iff-add*)
have *1019*: (*nlength σ* − *enat* (*Suc 0*) − *enat ia*) + *Suc ia* = *nlength σ*
by (*simp add*: *10183 10185*)
have *1020*: *nlast* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*) = *nlength σ*
by (*metis 1017 1019 nlast-nmap plus-enat-simps*(*1*))
have *1021*: *nlast* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) = *nlength σ*
using *1020* **by** *force*
have *1022*: *nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0* = *0*
by *simp*
have *1023*: *0* < *nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *1*

using *zero-enat-def* **by** *force*
have *1024*: *nfinite* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
by (*simp add*: *1017*)
have *1025*: *nidx* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
using *1017 nidx-Ncons-shift* **by** *blast*
have *1028*: (*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*))) =
(*nsubn σ 0* (*Suc ia*))
using *1017 zero-enat-def* **by** *fastforce*
have *1029*: ((*nsubn σ 0* (*Suc ia*)) ⊨ (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) ⟷
(∃ *i*<*length F-now*. *nsubn σ 0* (*Suc ia*) ⊨ *F-now* ! *i iand next F-next* ! *i*)
using *big-ior-map2-defs*[*of F-now F-next* (*nsubn σ 0* (*Suc ia*)) (λ *x y*. *x iand* (*next y*))]
assms **by** *blast*
have *1030*: *nsubn σ 0* (*Suc ia*) ⊨ *F-now* ! *i*
using *1016*
by (*metis 1012 1013 assms*(*3*) *ndropn-0 nsubn-def1 ntaken-nfirst state-pitl-defs state-pitl-list-defs*)
have *1031*: *nsubn σ 0* (*Suc ia*) ⊨ *next F-next* ! *i*
using *1016 ndropn-nsubn*[*of Suc ia σ Suc 0 0*]
by (*metis 1019 One-nat-def add-diff-cancel-left′ enat-le-plus-same*(*2*) *ileI1 le-add1*
nsubn-def1 nsubn-nlength-gr-one one-eSuc plus-1-eq-Suc semantics-pitl.simps(*4*) *zero-less-Suc*)
have *1034*: (*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*)) ⊨
((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))))
using *1012 1028 1029 1030 1031* **by** *auto*
have *1035*: ⋀*j*. *j*<*nlength l* ⟹
(*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *j*) = *nnth l j* + *Suc ia*
by *simp*
have *1036*: ⋀*j*. *j*<*nlength l* ⟹
(*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc j*)) = *nnth l* (*Suc j*) + *Suc ia*
using *Suc-ile-eq* **by** *force*
have *1037*: ⋀*j*. *j*<*nlength l* ⟹
nnth l j + *Suc ia* < *nnth l* (*Suc j*) + *Suc ia*
by (*metis 1017 add-strict-right-mono eSuc-enat ileI1 nidx-expand*)
have *1038*: ⋀*j*. *j*<*nlength l* ⟹
enat (*nnth l* (*Suc j*) + *Suc ia*) ≤ *nlength σ*
using *1017 nidx-expand* **by** *simp*
(*metis 1020 1025 1036 add-Suc-right eSuc-enat enat-ord-simps*(*1*) *ileI1 nfinite-nmap*
nidx-LCons-1 nidx-all-le-nlast nlength-nmap)
have *1039*: ⋀*j*. *j*<*nlength l* ⟹
Suc 0 + (*nnth l j* + *Suc ia*) ≤ *nnth l* (*Suc j*) + *Suc ia*
using *1037* **by** *fastforce*
have *1040*: ⋀*j*. *j*<*nlength l* ⟹
ndropn (*Suc 0*) (*nsubn σ* (*nnth l j* + *Suc ia*) (*nnth l* (*Suc j*) + *Suc ia*)) =
nsubn σ (*Suc 0* + (*nnth l j* + *Suc ia*)) (*nnth l* (*Suc j*) + *Suc ia*)
using *ndropn-nsubn*[*of - σ Suc 0*] **using** *1038 1039* **by** *presburger*
have *1041*: ⋀*j*. *j*<*nlength l* ⟹
(*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l j*) (*nnth l* (*Suc j*))) =
nsubn σ ((*nnth l j*) + *Suc ia*) ((*nnth l* (*Suc j*)) + *Suc ia*)
using *nsubn-ndropn*[*of - - σ Suc ia*]
by (*metis 1037 One-nat-def add-less-cancel-right ndropn-ndropn plus-1-eq-Suc*)

have *1280*: (*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*)) ⊨ *F*)
using *1034 assms*(*1*) **by** *simp*
have *1281*: (∀ *i*. *enat i* < *nlength* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *1017 1036 1041* **by** *auto*
have *1290*: (∀ *i*. *enat i* < *nlength* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*)) ⟷
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*)) ⊨ *F*) ∧
(∀ *i*. *enat i* < *nlength* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *forall-ncons-split*[*of* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
λ *x y*. (*nsubn σ x y* ⊨ *F*)] **by** (*simp add*: *1017*)
have *1300*: (∀ *i*. *enat i* < *nlength* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *1280 1281 1290* **by** *blast*
have *1400*: ∃ *l*. *nnth l 0* = *0* ∧ *nfinite l* ∧ *nidx l* ∧ *enat* (*nlast l*) = *nlength σ* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*))
by (*meson 1017 1021 1022 1025 1300 nfinite-NConsI nfinite-nmap*)
show *?thesis* **using** *1400* **by** *simp*
qed
show *?thesis*
using *1001 1002 1009* **by** *blast*
qed
show *?thesis*
using *1000 2* **by** *blast*
qed

lemma *GNF-omega-step*:
assumes ⊢ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-pitl F-e
state-pitl-list F-now
length F-now = *length F-next*
shows (*σ* ⊨ *omega F*) ⟷
(*σ* ⊨ (*ifalse iand empty*) *ior*
(*big-ior* (*map2* (λ *x y*. *x iand next y*) *F-now* (*map* (λ*x*. *x schop* (*omega F*)) *F-next*))))
(**is** *?lhs* ⟷ *?rhs*)
proof −
have *1*: (*σ* ⊨ *omega F*) ⟷
(∃ *l*. ¬*nfinite l* ∧ *nnth l 0* = *0* ∧ *nidx l* ∧
(∀ *i*. (*nnth l i*) ≤ *nlength σ*) ∧
(∀ *i*. (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*)))
by *simp*

have *2*: *?lhs* $\Longrightarrow$ *?rhs*
proof −
assume *l*: *?lhs*
obtain *l* **where** *3*: $\neg$*nfinite l* $\wedge$ *nnth l 0* = *0* $\wedge$ *nidx l* $\wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models F))$
using *l 1* **by** *blast*
have *6*: $\neg$*nfinite* σ
using *3 infinite-nidx-imp-infinite-interval* **by** *blast*
have *7*: *0* < *nlength* σ
using *6 enat-defs(1) nfinite-conv-nlength-enat* **by** *fastforce*
have *8*: (*nsubn* σ (*nnth l 0*) (*nnth l* (*Suc 0*)) $\models$ *F*)
using *3* **by** *blast*
have *9*: (*nsubn* σ (*nnth l 0*) (*nnth l* (*Suc 0*))) = (*ntaken* (*nnth l* (*Suc 0*)) σ)
using *3* **by** (*simp add*: *nsubn-def1*)
have *10*: (*ntaken* (*nnth l* (*Suc 0*)) σ) $\models$ *F*
using *9 8* **by** *auto*
have *11*: (*ntaken* (*nnth l* (*Suc 0*)) σ) $\models$ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y. x iand* (*next y*)) *F-now F-next*)))
using *10 assms(1) itl-ieq* **by** *blast*
have *12*: (*ntaken* (*nnth l* (*Suc 0*)) σ) $\models$ (*big-ior* (*map2* (λ *x y. x iand* (*next y*)) *F-now F-next*))
by (*metis 11 3 9 One-nat-def empty-defs enat-defs(2) iand-defs iless-eSuc0 ior-defs less-numeral-extra(1) less-numeral-extra(3) nidx-gr-first nlength-eq-enat-nfiniteD not-less nsubn-nlength-gr-one one-eSuc zero-enat-def*)
have *13*: *0* < *nlength* (*ntaken* (*nnth l* (*Suc 0*)) σ)
by (*metis 3 9 One-nat-def iless-eSuc0 less-numeral-extra(1) linorder-not-less nidx-gr-first nlength-eq-enat-nfiniteD nsubn-nlength-gr-one one-eSuc one-enat-def zero-enat-def*)
have *14*: (*nlength* (*ntaken* (*nnth l* (*Suc 0*)) σ) > *0* $\wedge$
$(\exists\, i{<}length\ F\text{-}now.\ ((NNil\ (nfirst\ (ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma))) \models F\text{-}now\ !i\ ) \wedge$
$((ndropn\ 1\ (ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma)) \models F\text{-}next\ !i))\ )$
using *GNF-defs*[*of F F-e F-now F-next* (*ntaken* (*nnth l* (*Suc 0*)) σ)] *12 13*
by (*metis 10 assms(1) assms(2) assms(3) assms(4) order-less-irrefl*)
obtain *i* **where** *15*: *i*<*length F-now* $\wedge$ ((*NNil* (*nfirst* (*ntaken* (*nnth l* (*Suc 0*)) σ))) $\models$ *F-now* !*i*) $\wedge$
((*ndropn 1* (*ntaken* (*nnth l* (*Suc 0*)) σ)) $\models$ *F-next* !*i*)
using *14* **by** *blast*
have *19*: ((*ndropn 1* σ) $\models$ *F-next* !*i schop* (*omega F*)) $\longleftrightarrow$
$(\exists\, ia.\ enat\ ia \leq nlength\ \sigma - enat\ (Suc\ 0) \wedge$
$(ntaken\ ia\ (ndropn\ (Suc\ 0)\ \sigma) \models F\text{-}next\ !\ i) \wedge$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ (Suc\ 0) - enat\ ia) \wedge$
$(\forall\, i.\ nsubn\ (ndropn\ ia\ (ndropn\ (Suc\ 0)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models F)))$
by *simp*
have *191*: *enat* ((*nnth l* (*Suc 0*))−*1*) ≤ *nlength* σ − *enat* (*Suc 0*)
by (*metis 3 One-nat-def enat-minus-mono1 idiff-enat-enat*)
have *192*: *ntaken* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) σ) $\models$ *F-next* ! *i*
using *13 15 191 3 7*
by (*simp add*: *enat-0-iff(1) ntaken-ndropn-swap*)
have *193*: *nnth* (*nmap* (λ*x. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *0* = *0*
using *enat-defs(1)* **by** *auto*
have *194*: $\neg$*nfinite* (*nmap* (λ*x. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))

by (*simp add*: *3*)
have *195*: *nidx* (*ndropn 1 l*)
by (*metis 3 One-nat-def Suc-ile-eq dual-order.order-iff-strict enat-defs*(*1*)
nfinite-conv-nlength-enat nidx-ndropn zero-le)
have *196*: *nnth* (*ndropn 1 l*) *0* = (*nnth l* (*Suc 0*))
by *simp*
have *197*: *nidx* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
using *nidx-shiftm*[*of* (*ndropn 1 l*) (*nnth l* (*Suc 0*))]
by (*metis 195 196 One-nat-def*)
have *198*: (∀ *i*. *enat* (*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *i*) ≤ *nlength σ* − *enat* (*Suc 0*) − *enat* ((*nnth l* (*Suc 0*))−*1*))
by (*metis 6 ndropn-nlength nfinite-ndropn nfinite-ntaken nle-le ntaken-all*)
have *202*: *nlength* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) = *nlength l* − (*Suc 0*)
by *force*
have *203*: (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*)) = (*ndropn* (*nnth l* (*Suc 0*)) *σ*)
by (*metis 3 One-nat-def Suc-pred ileI1 less-numeral-extra*(*1*) *ndropn-ndropn nidx-gr-first*
nlength-eq-enat-nfiniteD one-eSuc one-enat-def plus-1-eq-Suc zero-enat-def zero-less-iff-neq-zero)
have *204*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j* =
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
by *simp*
have *2041*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*) =
nnth l ((*Suc j*) + (*Suc 0*)) − (*nnth l* (*Suc 0*))
by (*metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap plus-1-eq-Suc*)
have *205*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)) < *nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
using *204 197* **unfolding** *nidx-expand* **apply** *auto*
by (*metis 204 One-nat-def add.commute eSuc-enat ileI1 plus-1-eq-Suc*)
have *206*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*))
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j*)
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*))) =
(*nsubn* (*ndropn* (*nnth l* (*Suc 0*)) *σ*)
(*nnth l* (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))
(*nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*))))
using *203 204 2041* **by** *simp*
have *207*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* (*ndropn* (*nnth l* (*Suc 0*)) *σ*)
(*nnth l* (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))
(*nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))) =
(*nsubn σ* (*nnth l* (*Suc j*)) (*nnth l* (*Suc* (*Suc j*))))
using *nsubn-ndropn*[*of* - - *σ* (*nnth l* (*Suc 0*))] **by** *simp*
(*metis* (*no-types*, *lifting*) *197 202 204 205 add.commute add.right-neutral*
bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first
order-less-imp-le plus-1-eq-Suc zero-less-diff)
have *208*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn σ* (*nnth l* (*Suc j*)) (*nnth l* (*Suc* (*Suc j*)))) ⊨ *F*
using *3* **by** *blast*

```
  have 209: (∀ i. enat i < nlength (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) ⟶
            (nsubn (ndropn ((nnth l (Suc 0))−1 ) (ndropn (Suc 0) σ))
             (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
             (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i)) ⊨ F))
        using 202 206 207 208 by presburger
  have 2090: nlength (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) = ∞
     by (metis 194 enat2-cases nlength-eq-enat-nfiniteD)
  have 2091: (∀ i.
            (nsubn (ndropn ((nnth l (Suc 0))−1 ) (ndropn (Suc 0) σ))
             (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
             (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i)) ⊨ F))
    using 2090 209
    using enat-ord-code(4) by presburger
  let ?ia = ((nnth l (Suc 0))−1 )
  let ?l = (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l))
  have 210: ( enat ?ia ≤ nlength σ − enat (Suc 0) ∧
            (ntaken ?ia (ndropn (Suc 0) σ) ⊨ F-next ! i) ∧
             (¬nfinite ?l ∧
              nnth ?l 0 = 0 ∧
              nidx ?l ∧
              (∀ i. enat (nnth ?l i) ≤ nlength σ − enat (Suc 0) − enat ?ia) ∧
              (∀ i.
                (nsubn (ndropn ?ia (ndropn (Suc 0) σ))
                   (nnth ?l i)
                   (nnth ?l (Suc i)) ⊨ F))
             )
            )
       using 191 192 193 194 197 198 2091 by blast
  have 20: ((ndropn 1 σ) ⊨ F-next !i schop (omega F))
      using 210 19 by blast
  have 21: σ ⊨ F-now ! i
      by (metis 15 assms(3) ntaken-nfirst state-pitl-defs state-pitl-list-defs)
  have 22: length F-now = length (map (λx.  x schop omega F  ) F-next)
      by (simp add: assms(4))
  have 23:  (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx.  x schop omega F  ) F-next))) ⟷
            (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop omega F  ) F-next ! i)
     using  big-ior-map2-defs[of F-now  (map (λx. x schop (omega F)) F-next) σ (λ x y. x iand next y)]

       22 by auto
    have 24: ( σ ⊨ F-now ! i iand next map (λx.  x schop omega F  ) F-next ! i)
       by (metis 15 20 21 7 assms(4) iand-defs ileI1 nth-map one-eSuc semantics-pitl.simps(4))
    show ?thesis using 15 23 24 ior-defs by blast
  qed
have 1000: ?rhs ⟹ ?lhs
 proof −
  assume r: ?rhs
  have 1001:  (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx.  x schop omega F  ) F-next)))
     using r by simp
  have 1010: length F-now = length (map (λx.  x schop omega F  ) F-next)
      by (simp add: assms(4))
```

have *1011*: ($\exists$ *i*<*length F-now*. $\sigma \models$ *F-now* ! *i iand next map* (λ*x*. *x schop omega F*) *F-next* ! *i*)
using *1001 1010 big-ior-map2-defs*[*of F-now* (*map* (λ*x*. *x schop* (*omega F*)) *F-next*) σ (λ *x y*. *x iand next y*)]
by *blast*
obtain *i* **where** *1012*: *i*<*length F-now* $\wedge$ ($\sigma \models$ *F-now* ! *i iand next map* (λ*x*. *x schop omega F*) *F-next* ! *i*)
using *1011* **by** *blast*
have *1013*: ($\sigma \models$ *F-now* ! *i*)
using *1012* **by** *auto*
have *1014*: ((*ndropn 1* σ) $\models$ *F-next* !*i schop* (*omega F*))
using *1012 assms*(*4*) **by** *fastforce*
have *1015*: ((*ndropn 1* σ) $\models$ *F-next* !*i schop* (*omega F*)) $\longleftrightarrow$
($\exists$ *ia*. *enat ia* $\leq$ *nlength* σ $-$ *enat* (*Suc 0*) $\wedge$
(*ntaken ia* (*ndropn* (*Suc 0*) σ) $\models$ *F-next* ! *i*) $\wedge$
($\exists$ *l*. $\neg$ *nfinite l* $\wedge$
nnth l 0 = *0* $\wedge$ *nidx l* $\wedge$
($\forall$ *i*. *enat* (*nnth l i*) $\leq$ *nlength* σ $-$ *enat* (*Suc 0*) $-$ *enat ia*) $\wedge$
($\forall$ *i*. *nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) σ)) (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *F*)))
by *simp*
obtain *ia* **where** *1016*: *enat ia* $\leq$ *nlength* σ $-$ *enat* (*Suc 0*) $\wedge$
(*ntaken ia* (*ndropn* (*Suc 0*) σ) $\models$ *F-next* ! *i*) $\wedge$
($\exists$ *l*. $\neg$ *nfinite l* $\wedge$
nnth l 0 = *0* $\wedge$ *nidx l* $\wedge$
($\forall$ *i*. *enat* (*nnth l i*) $\leq$ *nlength* σ $-$ *enat* (*Suc 0*) $-$ *enat ia*) $\wedge$
($\forall$ *i*. *nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) σ)) (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *F*))
using *1014 1015* **by** *blast*
obtain *l* **where** *1017*: $\neg$ *nfinite l* $\wedge$
nnth l 0 = *0* $\wedge$ *nidx l* $\wedge$
($\forall$ *i*. *enat* (*nnth l i*) $\leq$ *nlength* σ $-$ *enat* (*Suc 0*) $-$ *enat ia*) $\wedge$
($\forall$ *i*. *nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) σ)) (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *F*)
using *1016* **by** *blast*
have *1018*: $\neg$ *nfinite* σ
by (*metis 1016 infinite-nidx-imp-infinite-interval ndropn-nlength nfinite-ndropn-a*)
have *1022*: *nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0* = *0*
by *simp*
have *1023*: *0* < *nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *1*
using *zero-enat-def* **by** *force*
have *1024*: $\neg$*nfinite* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
by (*simp add*: *1017*)
have *1025*: *nidx* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
using *1017 nidx-Ncons-shift-alt* **by** *blast*
have *1026*: ($\forall$ *i*. (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) $\leq$ *nlength* σ)
by (*meson 1018 enat-ile nle-le nlength-eq-enat-nfiniteD*)
have *1028*: (*nsubn* σ (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*))) =
(*nsubn* σ *0* (*Suc ia*))
using *1017 zero-enat-def* **by** *fastforce*
have *1029*: ((*nsubn* σ *0* (*Suc ia*)) $\models$ (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) $\longleftrightarrow$
($\exists$ *i*<*length F-now*. *nsubn* σ *0* (*Suc ia*) $\models$ *F-now* ! *i iand next F-next* ! *i*)
using *big-ior-map2-defs*[*of F-now F-next* (*nsubn* σ *0* (*Suc ia*)) (λ *x y*. *x iand* (*next y*))]

assms **by** *blast*
have *1030*: *nsubn σ 0 (Suc ia) ⊨ F-now ! i*
using *1016*
by (*metis 1012 1013 assms(3) ndropn-0 nsubn-def1 ntaken-nfirst state-pitl-defs state-pitl-list-defs*)
have *1031*: *nsubn σ 0 (Suc ia) ⊨ next F-next ! i*
using *1016 ndropn-nsubn*[*of Suc ia σ Suc 0 0*]
by (*metis 1022 1023 1026 1028 One-nat-def add.commute diff-zero enat-min-eq enat-ord-simps(1) ileI1 leD ndropn-0 nsubn-def1 nsubn-nlength-gr-one ntaken-ndropn-swap one-eSuc one-enat-def plus-1-eq-Suc semantics-pitl.simps(4) zero-less-iff-neq-zero*)
have *1034*: (*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*)) ⊨
((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))))
using *1012 1028 1029 1030 1031* **by** *auto*
have *1035*: ⋀*j*. *j*<*nlength l* ⟹
(*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *j*) = *nnth l j* + *Suc ia*
by *simp*
have *1036*: ⋀*j*. *j*<*nlength l* ⟹
(*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc j*)) = *nnth l* (*Suc j*) + *Suc ia*
using *Suc-ile-eq* **by** *force*
have *1037*: ⋀*j*. *j*<*nlength l* ⟹
nnth l j + *Suc ia* < *nnth l* (*Suc j*) + *Suc ia*
by (*metis 1017 add-strict-right-mono eSuc-enat ileI1 nidx-expand*)
have *1038*: ⋀*j*. *j*<*nlength l* ⟹
enat (*nnth l* (*Suc j*) + *Suc ia*) ≤ *nlength σ*
using *1017 nidx-expand* **by** *simp*
(*meson 1018 enat-ile nle-le nlength-eq-enat-nfiniteD*)
have *1039*: ⋀*j*. *j*<*nlength l* ⟹
Suc 0 + (*nnth l j* + *Suc ia*) ≤ *nnth l* (*Suc j*) + *Suc ia*
using *1037* **by** *fastforce*
have *1040*: ⋀*j*. *j*<*nlength l* ⟹
ndropn (*Suc 0*) (*nsubn σ* (*nnth l j* + *Suc ia*) (*nnth l* (*Suc j*) + *Suc ia*)) =
nsubn σ (*Suc 0* + (*nnth l j* + *Suc ia*)) (*nnth l* (*Suc j*) + *Suc ia*)
using *ndropn-nsubn*[*of - σ Suc 0*] **using** *1038 1039* **by** *presburger*
have *1041*: ⋀*j*. *j*<*nlength l* ⟹
(*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l j*) (*nnth l* (*Suc j*))) =
nsubn σ ((*nnth l j*) + *Suc ia*) ((*nnth l* (*Suc j*)) + *Suc ia*)
using *nsubn-ndropn*[*of - - σ Suc ia*]
by (*metis 1037 One-nat-def add-less-cancel-right ndropn-ndropn plus-1-eq-Suc*)
have *1280*: (*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*)) ⊨ *F*)
using *1034 assms(1)* **by** *simp*
have *1281*: (∀ *i*. *enat i* < *nlength* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *1017 1036 1041*
by (*metis 1035 nlength-nmap*)
have *1290*: (∀ *i*. *enat i* < *nlength* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*)) ⟷
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc*

ia) *l*)) (*Suc 0*)) ⊨ *F*) ∧
(∀ *i*. *enat i* < *nlength* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *forall-ncons-split-alt*[*of* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
λ *x y*. (*nsubn σ x y* ⊨ *F*)]
by (*simp add*: *1017*)
have *1300*: (∀ *i*. *enat i* < *nlength* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *1280 1281 1290* **by** *blast*
have *1340*: *nlength* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) = ∞
by (*metis 1024 enat2-cases nlength-eq-enat-nfiniteD*)
have *1350*: (∀ *i*.
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *1300 1340 enat-ord-code*(*4*) **by** *presburger*
have *1400*: (∃ (*l*:: *nat nellist*).
¬*nfinite l* ∧ (*nnth l 0*) = *0* ∧ *nidx l* ∧
(∀ *i*. (*nnth l i*) ≤ *nlength σ*) ∧
(∀ *i*. ((*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *F*))
)
by (*meson 1017 1022 1025 1026 1350 nfinite-NCons nfinite-nmap*)
show *?thesis* **using** *1400* **by** *simp*
qed
show *?thesis*
using *1000 2* **by** *blast*
qed

lemma *GNF-atom-base*:
⊢ (*atom p*) *ieqv* (*GNF* (*atom p*))
unfolding *GNF-def GNF-cons-def big-ior-def*
by (*auto*)
(*metis add-0 ileI1 linorder-not-less not-less-zero order-neq-le-trans plus-1-eSuc*(*2*))

lemma *GNF-ifalse-base*:
⊢ *ifalse ieqv* (*GNF ifalse*)
unfolding *GNF-def GNF-cons-def big-ior-def*
by *auto*

lemma *GNF-next-step*:
⊢ (*next f*) *ieqv* (*GNF* (*next f*))
unfolding *GNF-def GNF-cons-def big-ior-def*
by (*auto*)

lemma *exists-GNF-nondet*:

```
 ∃ ae a A. (state-pitl ae) ∧ (state-pitl-list a) ∧ length a = length A ∧
    (⊢ f ieqv (ae iand empty) ior (big-ior (map2 (λ x y. x iand next y) a A)))
proof (induction f)
case false-d
then show ?case
  proof −
   have f1: ⊢ ifalse ieqv (GNF ifalse)
     using GNF-ifalse-base by auto
   have f2: state-pitl-list (GNF-state ifalse)
     unfolding  state-pitl-list-def by (simp add: inot-d-def itrue-d-def)
   have f3: length (GNF-state ifalse) = length (GNF-next ifalse)
     by simp
   show ?thesis
   by (metis f1 f2 f3 GNF-cons-def GNF-def GNF-empty.simps(1) state-pitl.simps(1))
  qed
next
case (atom-d x)
then show ?case
  proof −
   have a1: ⊢(atom x) ieqv (GNF (atom x))
     using GNF-atom-base by blast
   have a2: state-pitl-list (GNF-state (atom x))
     unfolding  state-pitl-list-def by simp
   have a3: length (GNF-state (atom x)) = length (GNF-next (atom x))
    by simp
   show ?thesis
   by (metis GNF-cons-def GNF-def GNF-empty.simps(2) a1 a2 a3 state-pitl.simps(2))
  qed
next
case (iimp-d f1 f2)
then show ?case
 proof −
  obtain ae1 a1 A1 where i1: state-pitl ae1 ∧ state-pitl-list a1 ∧ length a1 = length A1 ∧
     (⊢ f1 ieqv ae1 iand empty ior big-ior (map2 (λx y. x iand next y) a1 A1))
     using iimp-d.IH(1) by blast
  obtain ae2 a2 A2 where i2: state-pitl ae2 ∧ state-pitl-list a2 ∧ length a2 = length A2 ∧
     (⊢ f2 ieqv ae2 iand empty ior big-ior (map2 (λx y. x iand next y) a2 A2))
     using iimp-d.IH(2) by blast
  have i3: ⊢ (f1 iimp f2) ieqv ( ( (inot ae1) ior ae2) iand empty) ior
            (big-ior
             (map2 (λ x y. x iand (next y))
                  ((add-to-now a1 ) @ a2)
                  ((map (λx. (inot x)) (add-to-next A1 )) @ A2) ))
      using GNF-iimp-step[of f1 ae1 a1 A1 f2 ae2 a2 A2] i1 i2 itl-ieq by blast
  have i4: length ((add-to-now a1 ) @ a2) = length ((map (λx. (inot x)) (add-to-next A1 )) @ A2)
    by (simp add: add-to-next-def add-to-now-def i1 i2 length-add-to-now-next-gen)
  have i5: state-pitl-list ((add-to-now a1 ) @ a2)
    by (simp add: add-to-now-def i1 i2 state-pitl-foldr-add-to-now-single-inv-a state-pitl-list-append)
   show ?thesis
   by (metis i1 i2 i3 i4 i5 inot-d-def ior-d-def state-pitl.simps(1) state-pitl.simps(3))
```

```
  qed
next
case (next-d f)
then show ?case
  proof −
   obtain ae a A where n1:  state-pitl ae ∧ state-pitl-list a ∧ length a = length A ∧
     (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
      using next-d.IH by blast
   have n2: ⊢ (next f) ieqv (GNF (next f))
    using GNF-next-step by blast
   have n3: state-pitl-list (GNF-state (next f))
     by (simp add: inot-d-def itrue-d-def state-pitl-list-defs)
   have n4: length (GNF-state (next f)) = length (GNF-next (next f))
     by simp
   show ?thesis
   by (metis GNF-cons-def GNF-def GNF-empty.simps(4) n2 n3 n4 state-pitl.simps(1))
  qed
next
case (chop-d f1 f2)
then show ?case
  proof −
   obtain ae1 a1 A1 where c1: state-pitl ae1 ∧ state-pitl-list a1 ∧ length a1 = length A1 ∧
     (⊢ f1 ieqv ae1 iand empty ior big-ior (map2 (λx y. x iand next y) a1 A1) )
    using chop-d.IH(1) by blast
   obtain ae2 a2 A2 where c2: state-pitl ae2 ∧ state-pitl-list a2 ∧ length a2 = length A2 ∧
     (⊢ f2 ieqv ae2 iand empty ior big-ior (map2 (λx y. x iand next y) a2 A2) )
    using chop-d.IH(2) by blast
   have c3: ⊢ f1 schop f2 ieqv (ae1 iand ae2) iand empty ior
              (big-ior (map2 (λ x y. x iand next y)
                       (a1 @ (map (λx. x iand ae1) a2))
                       ((map (λx. x schop f2) A1) @  A2)))
     using GNF-schop-step[of f1 ae1 a1 A1 f2 ae2 a2 A2] c1 c2 itl-ieq by blast
   have c4: length (a1 @ (map (λx. x iand ae1) a2)) = length ((map (λx. x schop f2) A1) @  A2)
      by (simp add: c1 c2)
   have c5: state-pitl-list (a1 @ (map (λx. x iand ae1) a2))
    by (simp add: c1 c2 state-pitl-list-append state-pitl-list-map-alt)
   have c6:  state-pitl (ae1 iand ae2)
       by (simp add: c1 c2 iand-d-def inot-d-def ior-d-def)
   show ?thesis using c1 c2 c3 c4 c5 c6 by blast
  qed
next
case (schopstar-d f)
then show ?case
  proof −
   obtain ae a A where s1:  state-pitl ae ∧ state-pitl-list a ∧ length a = length A ∧
     (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
     using schopstar-d.IH by blast
   have s2: ⊢ schopstar f ieqv (itrue iand empty) ior
              (big-ior (map2 (λ x y. x iand next y) a (map (λx. x schop (schopstar f)) A)))
     using GNF-schopstar-step[of f ae a A] s1 using itl-ieq by blast
```

```
    have s3: length a = length (map (λx. x schop (schopstar f)) A)
      by (simp add: s1)
    have s4: state-pitl itrue
      by (simp add: inot-d-def itrue-d-def)
    show ?thesis
    by (metis  s1 s2 s3 s4)
   qed
next
case (omega-d f)
then show ?case
  proof −
   obtain ae a A where o1:  state-pitl ae ∧ state-pitl-list a ∧ length a = length A ∧
     (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
     using omega-d.IH by blast
   have o2: ⊢ omega f ieqv (ifalse iand empty) ior
              (big-ior (map2 (λ x y. x iand next y) a (map (λx. x schop (omega f)) A)))
     using GNF-omega-step[of f ae a A] o1 using itl-ieq by blast
   have o3: length a = length (map (λx. x schop (omega f)) A)
     by (simp add: o1)
   show ?thesis
   using o1 o2 o3 state-pitl.simps(1) by blast
  qed
qed

lemma state-pitl-GNF-empty:
 state-pitl (GNF-empty f)
proof (induction f)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case by simp
next
case (iimp-d f1 f2)
then show ?case  apply simp  unfolding inot-d-def ior-d-def by simp
next
case (next-d f)
then show ?case by simp
next
case (chop-d f1 f2)
then show ?case apply simp unfolding iand-d-def inot-d-def ior-d-def by simp
next
case (schopstar-d f)
then show ?case apply simp unfolding itrue-d-def inot-d-def by simp
next
case (omega-d f)
then show ?case by simp
qed

lemma state-pitl-list-GNF-state:
```

state-pitl-list (*GNF-state f*)
proof (*induction f*)
case *false-d*
then show *?case* **unfolding** *state-pitl-list-def* **apply** *simp* **unfolding** *itrue-d-def inot-d-def* **by** *simp*
next
case (*atom-d x*)
then show *?case* **unfolding** *state-pitl-list-def* **by** *simp*
next
case (*iimp-d f1 f2*)
then show *?case* **unfolding** *state-pitl-list-def* **apply** *simp*
by (*metis add-to-now-def state-pitl-foldr-add-to-now-single-inv-a state-pitl-list-def*)
next
case (*next-d f*)
then show *?case* **unfolding** *state-pitl-list-def* **apply** *simp* **unfolding** *itrue-d-def inot-d-def* **by** *simp*
next
case (*chop-d f1 f2*)
then show *?case* **unfolding** *state-pitl-list-def* **apply** *simp*
by (*metis* (*full-types*) *state-pitl-GNF-empty state-pitl-list-def state-pitl-list-map-alt*)
next
case (*schopstar-d f*)
then show *?case* **unfolding** *state-pitl-list-def* **by** *simp*
next
case (*omega-d f*)
then show *?case* **unfolding** *state-pitl-list-def* **by** *simp*
qed

lemma *length-GNF-state-eq-length-GNF-next*:
length (*GNF-state f*) = *length* (*GNF-next f*)
proof (*induction f*)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case* **by** *simp*
next
case (*iimp-d f1 f2*)
then show *?case*
by (*simp add*: *add-to-next-def add-to-now-def length-add-to-now-next-gen*)
next
case (*next-d f*)
then show *?case* **by** *simp*
next
case (*chop-d f1 f2*)
then show *?case* **by** *simp*
next
case (*schopstar-d f*)
then show *?case* **by** *simp*
next
case (*omega-d f*)
then show *?case* **by** *simp*

qed

lemma *ieqv-GNF*:
⊢ *f ieqv GNF f*
proof (*induction f*)
case *false-d*
then show *?case*
using *GNF-ifalse-base* **by** *auto*
next
case (*atom-d x*)
then show *?case* **using** *GNF-atom-base* **by** *blast*
next
case (*iimp-d f1 f2*)
then show *?case*
proof −
have *i1*: ⊢ *f1 ieqv GNF-empty f1 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f1*) (*GNF-next f1*))
using *iimp-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *i2*: ⊢ *f2 ieqv GNF-empty f2 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f2*) (*GNF-next f2*))
using *iimp-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *i3*: *state-pitl* (*GNF-empty f1*)
by (*simp add*: *state-pitl-GNF-empty*)
have *i4*: *state-pitl-list* (*GNF-state f1*)
by (*simp add*: *state-pitl-list-GNF-state*)
have *i5*: *state-pitl* (*GNF-empty f2*)
by (*simp add*: *state-pitl-GNF-empty*)
have *i6*: *state-pitl-list* (*GNF-state f2*)
by (*simp add*: *state-pitl-list-GNF-state*)
have *i7*: *length* (*GNF-state f1*) = *length* (*GNF-next f1*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
have *i8*: *length* (*GNF-state f2*) = *length* (*GNF-next f2*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
show *?thesis* **using**
GNF-iimp-step[*of f1 GNF-empty f1* (*GNF-state f1*) (*GNF-next f1*)
f2 GNF-empty f2 (*GNF-state f2*) (*GNF-next f2*)]
using *GNF-cons-def GNF-def i1 i2 i3 i4 i5 i6 i7 i8*
by (*metis GNF-empty.simps*(*3*) *GNF-next.simps*(*3*) *GNF-state.simps*(*3*) *itl-ieq*)
qed
next
case (*next-d f*)
then show *?case* **using** *GNF-next-step* **by** *blast*
next
case (*chop-d f1 f2*)
then show *?case*
proof −
have *c1*: ⊢ *f1 ieqv GNF-empty f1 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f1*) (*GNF-next f1*))
using *chop-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *c2*: ⊢ *f2 ieqv GNF-empty f2 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f2*)

```
(GNF-next f2))
    using chop-d by (simp add: GNF-cons-def GNF-def)
  have c3: state-pitl (GNF-empty f1)
    by (simp add: state-pitl-GNF-empty)
  have c4: state-pitl-list (GNF-state f1)
    by (simp add: state-pitl-list-GNF-state)
  have c5: state-pitl (GNF-empty f2)
    by (simp add: state-pitl-GNF-empty)
  have c6: state-pitl-list (GNF-state f2)
    by (simp add: state-pitl-list-GNF-state)
  have c7: length (GNF-state f1) = length (GNF-next f1)
    by (simp add: length-GNF-state-eq-length-GNF-next)
  have c8: length (GNF-state f2) = length (GNF-next f2)
    by (simp add: length-GNF-state-eq-length-GNF-next)
  show ?thesis using
   GNF-schop-step[of f1 GNF-empty f1 (GNF-state f1) (GNF-next f1)
                  f2 GNF-empty f2 (GNF-state f2) (GNF-next f2)]
   unfolding GNF-def GNF-cons-def using c1 c2 c3 c4 c5 c6 c7 c8 by fastforce
  qed
next
case (schopstar-d f)
then show ?case
  proof −
    have s1: ⊢ f ieqv GNF-empty f iand empty ior big-ior (map2 (λx y. x iand next y) (GNF-state f)
(GNF-next f))
    using schopstar-d by (simp add: GNF-cons-def GNF-def)
   have s2: state-pitl (GNF-empty f)
    by (simp add: state-pitl-GNF-empty)
   have s3: state-pitl-list (GNF-state f)
    by (simp add: state-pitl-list-GNF-state)
   have s4: length (GNF-state f) = length (GNF-next f)
    by (simp add: length-GNF-state-eq-length-GNF-next)
   show ?thesis using
    GNF-schopstar-step[of f GNF-empty f (GNF-state f) (GNF-next f)]
     unfolding GNF-def GNF-cons-def using s1 s2 s3 s4 by fastforce
  qed
next
case (omega-d f)
then show ?case
 proof −
    have o1: ⊢ f ieqv GNF-empty f iand empty ior big-ior (map2 (λx y. x iand next y) (GNF-state f)
(GNF-next f))
    using omega-d by (simp add: GNF-cons-def GNF-def)
   have o2: state-pitl (GNF-empty f)
    by (simp add: state-pitl-GNF-empty)
   have o3: state-pitl-list (GNF-state f)
    by (simp add: state-pitl-list-GNF-state)
   have o4: length (GNF-state f) = length (GNF-next f)
    by (simp add: length-GNF-state-eq-length-GNF-next)
   show ?thesis using
```

GNF-omega-step[*of f GNF-empty f* (*GNF-state f*) (*GNF-next f*)]
unfolding *GNF-def GNF-cons-def* **using** *o1 o2 o3 o4* **by** *fastforce*
qed
qed

lemma *exist-GNF-det*:
GNF-prop f
proof −
have *1*: ∃ *ae a A*. (*state-pitl ae*) ∧ (*state-pitl-list a*) ∧ *length a* = *length A* ∧
(⊢ *f ieqv* (*ae iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand next y*) *a A*)))
using *exists-GNF-nondet* **by** *blast*
obtain *ae a A* **where** *2*: (*state-pitl ae*) ∧ (*state-pitl-list a*) ∧ *length a* = *length A* ∧
(⊢ *f ieqv* (*ae iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand next y*) *a A*)))
using *1* **by** *blast*
have *3*: ⊢ (*big-ior* (*map2* (λ *x y*. *x iand next y*) *a A*)) *ieqv*
big-ior (*map2* (λ *x y*. *x iand* (*next y*))
(*foldr* (*add-to-now-single*) *a* [*itrue*])
(*foldr* (*add-to-next-single*) *A* [*ifalse*]))
using *2 det-big-ior-ieqv-nondet-big-ior-a* **by** *auto*
have *4*: ⊢ *f ieqv* (*ae iand empty*) *ior*
big-ior (*map2* (λ *x y*. *x iand* (*next y*))
(*foldr* (*add-to-now-single*) *a* [*itrue*])
(*foldr* (*add-to-next-single*) *A* [*ifalse*]))
using *2 3* **by** *fastforce*
have *5*: *state-pitl-list* (*foldr* (*add-to-now-single*) *a* [*itrue*])
by (*simp add*: *2 state-pitl-foldr-add-to-now-single-inv-a*)
have *6*: *length* (*foldr* (*add-to-now-single*) *a* [*itrue*]) =
length (*foldr* (*add-to-next-single*) *A* [*ifalse*])
by (*simp add*: *2 length-add-to-now-next-gen*)
have *7*: *full-system* (*foldr* (*add-to-now-single*) *a* [*itrue*])
using *big-ior-foldr-add-to-now-single-inv-a full-system-def mutex-foldr-add-to-now-single-inv-a* **by** *blast*
show *?thesis* **unfolding** *GNF-prop-def GNF-cons-def*
using *2 4 5 6 7* **by** *blast*
qed

lemma *state-pitl-list-GNF-det-state*:
state-pitl-list (*foldr* (*add-to-now-single*) (*GNF-state f*) [*itrue*])
by (*simp add*: *state-pitl-foldr-add-to-now-single-inv-a state-pitl-list-GNF-state*)

lemma *full-system-GNF-det-state*:
full-system (*foldr* (*add-to-now-single*) (*GNF-state f*) [*itrue*])
using *big-ior-foldr-add-to-now-single-inv-a full-system-def*
mutex-foldr-add-to-now-single-inv-a **by** *blast*

lemma *ieqv-GNF-det*:
⊢ *f ieqv GNF-det f*

using *ieqv-GNF*[*of f*]
unfolding *GNF-def GNF-det-def GNF-cons-def*
using *det-big-ior-ieqv-nondet-big-ior-a length-GNF-state-eq-length-GNF-next* **by** *auto*

end

Chapter 5

Deep embedding of QPITL

theory *QPITL-Deep* **imports**
NELList-Extras HOL−Library.FSet
Update
begin

This theory is a deep embedding of Quantified Propositional Interval Temporal Logic (QPITL). It provides the syntax and semantics.

5.1 Syntax

5.1.1 Primitive formulae

datatype *qpitl* =

false-d	(*ifalse*)
| *atom-d nat*	(($-) [*100*] *99*)
| *iimp-d qpitl qpitl*	((- *iimp* -) [*26*,*25*] *25*)
| *next-d qpitl*	((*next* -) [*88*] *87*)
| *chop-d qpitl qpitl*	((- *schop* -) [*84*,*84*] *83*)
| *schopstar-d qpitl*	((*schopstar* -) [*85*] *85*)
| *omega-d qpitl*	((*omega* -) [*85*] *85*)
| *exists-d nat qpitl*	(*Ex* -. - *10*)

5.1.2 state formula

fun *state-qpitl* :: *qpitl* ⇒ *bool*
where

state-qpitl (*ifalse*)	= *True*
| *state-qpitl* ($*n*)	= *True*
| *state-qpitl* (*f iimp g*)	= ((*state-qpitl f*) ∧ (*state-qpitl g*))
| *state-qpitl* (*next f*)	= *False*
| *state-qpitl* (*f schop g*)	= *False*
| *state-qpitl* (*schopstar f*)	= *False*
| *state-qpitl* (*omega f*)	= *False*
| *state-qpitl* (*Ex n. f*)	= *state-qpitl f*

5.1.3 Derived Boolean Operators

definition *inot-d* ((*inot* -) [*90*] *90*)
where
inot f ≡ *f iimp ifalse*

definition *itrue-d* (*itrue*)
where
itrue ≡ *inot ifalse*

definition *ior-d* ((- *ior* -) [*31*,*30*] *30*)
where
f ior g ≡ (*inot f*) *iimp g*

definition *iand-d* ((- *iand* -) [*36*,*35*] *35*)
where
f iand g ≡ *inot* (*inot f ior inot g*)

definition *ieqv-d* ((- *ieqv* -) [*21*,*20*] *20*)
where
f ieqv g ≡ ((*f iimp g*) *iand* (*g iimp f*))

5.1.4 more, empty, skip and wnext

definition *more-d* (*more*)
where
more ≡ *next itrue*

definition *empty-d* (*empty*)
where
empty ≡ *inot more*

definition *skip-d* (*skip*)
where
skip ≡ *next empty*

definition *wnext-d* ((*wnext* -) [*88*] *87*)
where
wnext f ≡ *inot* (*next* (*inot f*))

5.1.5 ifinite and inf

definition *finite-d* (*ifinite*)
where
ifinite ≡ *itrue schop empty*

definition *inf-d* (*inf*)
where
inf ≡ *inot ifinite*

5.1.6 weak chop

definition *chop-d* ((- ; -) [*84*,*84*] *83*)
where $f\ ;\ g \equiv (f\ schop\ g)\ ior\ (f\ iand\ inf)$

5.1.7 Box and Diamond Operators

definition *sometimes-d* ((*diamond* -) [*88*] *87*)
where
$diamond\ f \equiv itrue\ schop\ f$

definition *always-d* ((*box* -) [*88*] *87*)
where
$box\ f \equiv inot\ (diamond\ (inot\ f))$

definition *di-d* ((*di* -) [*88*] *87*)
where
$di\ f \equiv f\ ;\ itrue$

definition *bi-d* ((*bi* -) [*88*] *87*)
where
$bi\ f \equiv inot\ (di\ (inot\ f))$

definition *df-d* ((*df* -) [*88*] *87*)
where
$df\ f \equiv f\ schop\ itrue$

definition *bf-d* ((*bf* -) [*88*] *87*)
where
$bf\ f \equiv inot\ (df\ (inot\ f))$

definition *da-d* ((*da* -) [*88*] *87*)
where
$da\ f \equiv itrue\ schop\ (f\ ;\ itrue)$

definition *ba-d* ((*ba* -) [*88*] *87*)
where
$ba\ f \equiv inot\ (da\ (inot\ f))$

definition *sda-d* ((*sda* -) [*88*] *87*)
where
$sda\ f \equiv itrue\ schop\ (f\ schop\ itrue)$

definition *sba-d* ((*sba* -) [*88*] *87*)
where
$sba\ f \equiv inot\ (sda\ (inot\ f))$

5.1.8 Initial and Final Operators

definition *init-d* ((*init* -) [*88*] *87*)
where

init f ≡ ((*empty iand f*) *schop itrue*)

definition *fin-d* ((*fin* -) [*88*] *87*)
where
fin f ≡ *diamond* (*empty iand f*)

5.1.9 Forall

definition *forall-d* (*Fa* -. - *10*)
where
Fa v. f ≡ *inot* (*Ex v. inot f*)

5.1.10 bind

definition *sfxbnd-d* (*sfxbnd* - ≃ - : - *10*)
where *sfxbnd v* ≃ *f1* : *f2* ≡ (*Ex v. box*(*$v ieqv f1*) *iand f2*)

definition *pfxbnd-d* (*pfxbnd* - ≃ - : - *10*)
where *pfxbnd v* ≃ *f1* : *f2* ≡ (*Ex v. bf*((*fin $v*) *ieqv f1*) *iand f2*)

5.1.11 Big operations

definition *big-ior* :: *qpitl list* ⇒ *qpitl*
where *big-ior F* = *foldr* (λ *x y. x ior y*) *F ifalse*

definition *big-iand* :: *qpitl list* ⇒ *qpitl*
where *big-iand F* = *foldr* (λ *x y. x iand y*) *F itrue*

definition *state-qpitl-list* :: *qpitl list* ⇒ *bool*
where *state-qpitl-list L* = *foldr* (λ*x y. state-qpitl x* ∧ *y*) *L True*

5.2 Semantics

5.2.1 Semantics Primitive Operators

primrec *fvars* :: *qpitl* ⇒ *nat set*
where
fvars ifalse = {}
| *fvars* (*$n*) = {*n*}
| *fvars* (*f iimp g*) = (*fvars f*) ∪ (*fvars g*)
| *fvars* (*next f*) = *fvars f*
| *fvars* (*f schop g*) = (*fvars f*) ∪ (*fvars g*)
| *fvars* (*schopstar f*) = *fvars f*
| *fvars* (*omega f*) = *fvars f*
| *fvars* (*Ex n. f*) = (*fvars f*) − {*n*}

primrec *bvars* :: *qpitl* ⇒ *nat set*
where

$$
\begin{array}{ll}
& bvars\ ifalse = \{\} \\
| & bvars\ (\$n) = \{\} \\
| & bvars\ (f\ iimp\ g) = (bvars\ f) \cup (bvars\ g) \\
| & bvars\ (next\ f) = bvars\ f \\
| & bvars\ (f\ schop\ g) = (bvars\ f) \cup (bvars\ g) \\
| & bvars\ (schopstar\ f) = bvars\ f \\
| & bvars\ (omega\ f) = bvars\ f \\
| & bvars\ (Ex\ n.\ f) = (insert\ n\ (bvars\ f))
\end{array}
$$

primrec *is-free* :: *nat* ⇒ *qpitl* ⇒ *qpitl* ⇒ *bool*
where

$$
\begin{array}{ll}
& is\text{-}free\ x\ ft\ ifalse = True \\
| & is\text{-}free\ x\ ft\ (\$n) = True \\
| & is\text{-}free\ x\ ft\ (f\ iimp\ g) = ((is\text{-}free\ x\ ft\ f) \wedge (is\text{-}free\ x\ ft\ g)) \\
| & is\text{-}free\ x\ ft\ (next\ f) = (is\text{-}free\ x\ ft\ f) \\
| & is\text{-}free\ x\ ft\ (f\ schop\ g) = ((is\text{-}free\ x\ ft\ f) \wedge (is\text{-}free\ x\ ft\ g)) \\
| & is\text{-}free\ x\ ft\ (schopstar\ f) = (is\text{-}free\ x\ ft\ f) \\
| & is\text{-}free\ x\ ft\ (omega\ f) = (is\text{-}free\ x\ ft\ f) \\
| & is\text{-}free\ x\ ft\ (Ex\ n.\ f) = ((\ x \in fvars(Ex\ n.\ f) \longrightarrow n \notin fvars\ ft) \wedge is\text{-}free\ x\ ft\ f)
\end{array}
$$

lemma
$$
\begin{array}{l}
is\text{-}free\ x\ ft\ (sfxbnd\ v \simeq f1 : f2) = \\
((x \in fvars\ (box\ (\$v\ ieqv\ f1\ )\ iand\ f2) \wedge x \neq v \longrightarrow v \notin fvars\ ft) \wedge is\text{-}free\ x\ ft\ (box\ (\$v\ ieqv\ f1\ )\ iand\ f2))
\end{array}
$$
unfolding *sfxbnd-d-def*
by *simp*

lemma *is-free-iand*:
$$
\begin{array}{l}
is\text{-}free\ x\ ft\ (f1\ iand\ f2) = \\
(\ is\text{-}free\ x\ ft\ f1 \wedge is\text{-}free\ x\ ft\ f2)
\end{array}
$$
unfolding *iand-d-def inot-d-def ior-d-def* **by** *simp*

lemma *is-free-box*:
$$
\begin{array}{l}
is\text{-}free\ x\ ft\ (box\ (\$v\ ieqv\ f1\ )) = \\
is\text{-}free\ x\ ft\ f1
\end{array}
$$
unfolding *always-d-def sometimes-d-def itrue-d-def inot-d-def ieqv-d-def iand-d-def ior-d-def*
by *simp*

lemma *fvars-box*:
$$
fvars\ (box\ (\$v\ ieqv\ f1\ )\ iand\ f2) = \{v\} \cup fvars\ f1 \cup fvars\ f2
$$
unfolding *always-d-def sometimes-d-def itrue-d-def inot-d-def ieqv-d-def iand-d-def ior-d-def*
by *simp*

lemma
$$
\begin{array}{l}
((x \in fvars\ (box\ (\$v\ ieqv\ f1\ )\ iand\ f2) \wedge x \neq v \longrightarrow v \notin fvars\ ft) \wedge is\text{-}free\ x\ ft\ (box\ (\$v\ ieqv\ f1\ )\ iand\ f2)) \\
\longleftrightarrow \\
((x \in fvars\ f1 \cup fvars\ f2 \wedge x \neq v \longrightarrow v \notin fvars\ ft) \wedge is\text{-}free\ x\ ft\ f1 \wedge is\text{-}free\ x\ ft\ f2)
\end{array}
$$
using *fvars-box is-free-box is-free-iand* **by** *auto*

primrec *suform* :: *qpitl* ⇒ *nat* ⇒ *qpitl* ⇒ *qpitl*
where
  suform ifalse n h = *ifalse*
| *suform* ($*k*) *n h* = (*if k* = *n then h else* ($*k*))
| *suform* (*f iimp g*) *n h* = ((*suform f n h*) *iimp* (*suform g n h*))
| *suform* (*next f*) *n h* = (*next* (*suform f n h*))
| *suform* (*f schop g*) *n h* = ((*suform f n h*) *schop* (*suform g n h*))
| *suform* (*schopstar f*) *n h* = (*schopstar* (*suform f n h*))
| *suform* (*omega f*) *n h* = (*omega* (*suform f n h*))
| *suform* (*Ex k. f*) *n h* = (*Ex k.* (*suform f n h*))

primrec *rename* :: *qpitl* ⇒ *nat* ⇒ *nat* ⇒ *qpitl*
where
  rename ifalse n k = *ifalse*
| *rename* ($*m*) *n k* = (*if m* = *n then* $*k else* $*m*)
| *rename* (*f iimp g*) *n k* = ((*rename f n k*) *iimp* (*rename g n k*))
| *rename* (*next f*) *n k* = (*next* (*rename f n k*))
| *rename* (*f schop g*) *n k* = ((*rename f n k*) *schop* (*rename g n k*))
| *rename* (*schopstar f*) *n k* = (*schopstar* (*rename f n k*))
| *rename* (*omega f*) *n k* = (*omega* (*rename f n k*))
| *rename* (*Ex m. f*) *n k* = (*Ex* (*if m* = *n then k else m*). (*rename f n k*))

primrec *max-var* :: *qpitl* ⇒ *nat* ⇒ *nat*
where
  max-var ifalse m = *m*
| *max-var* ($*k*) *m* = (*if m* < *k then k else m*)
| *max-var* (*f iimp g*) *m* = (*if* (*max-var f m*) < (*max-var g m*) *then* (*max-var g m*) *else* (*max-var f m*))
| *max-var* (*next f*) *m* = *max-var f m*
| *max-var* (*f schop g*) *m* = (*if* (*max-var f m*) < (*max-var g m*) *then* (*max-var g m*) *else* (*max-var f m*))
| *max-var* (*schopstar f*) *m* = *max-var f m*
| *max-var* (*omega f*) *m* = *max-var f m*
| *max-var* (*Ex k. f*) *m* = (*if* (*max-var f m*) < *k then k else* (*max-var f m*))

fun *semantics-qpitl* :: [*interval*, *qpitl*] ⇒ *bool* ((- ⊨ -) [*80*,*10*] *10*)
where
  (*σ* ⊨ *ifalse*) = *False*
| (*σ* ⊨ $*p*) = (*p* |∈| (*nfirst σ*))
| (*σ* ⊨ *f iimp g*) = ((*σ* ⊨ *f*) ⟶ (*σ* ⊨ *g*))
| (*σ* ⊨ (*next f*)) = (*1* ≤ *nlength σ* ∧ ((*ndropn 1 σ*) ⊨ *f*))
| (*σ* ⊨ *f schop g*) =
  (∃ *i*. ((*i* ≤ (*nlength σ*)) ∧ ((*ntaken i σ*) ⊨ *f*) ∧ ((*ndropn i σ*) ⊨ *g*)))
| (*σ* ⊨ (*schopstar f*)) =
  (∃ (*l*:: *nat nellist*).
    (*nnth l 0*) = *0* ∧ *nfinite l* ∧ *nidx l* ∧
    (*enat* (*nlast l*)) = (*nlength σ*) ∧ *nfinite σ* ∧
    (∀ (*i*::*nat*) . ((*enat i*)< (*nlength l*)) ⟶

$$((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\ )\models f)\ )$$
$$)$$

$$|\ (\sigma \models (omega\ f)) =$$
$$(\exists (l::\ nat\ nellist).$$
$$\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$$
$$(\forall i.\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$$
$$(\forall i.\ (\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models f)\ )$$
$$)$$
$$|\ \ (\sigma \models (Ex\ n.\ f)\ ) = (\exists\ \ l.\ nlength\ \sigma = nlength\ l \wedge ((upd\ \sigma\ n\ l)\ \ \models f)\ )$$

lemma *not-fvar-upd*:
assumes $n \notin fvars\ f$
$nlength\ l = nlength\ \sigma$
shows $(\sigma \models f) = ((upd\ \sigma\ n\ l) \models f)$
using *assms*
proof (*induction f arbitrary*: $\sigma\ l$)
case *false-d*
then show *?case*
using *semantics-qpitl.simps*(*1*) **by** *presburger*
next
case (*atom-d x*)
then show *?case*
apply *simp*
by (*metis nfirst-eq-nnth-zero similar-defs similar-upd zero-enat-def zero-le*)
next
case (*iimp-d f1 f2*)
then show *?case* **by** *simp*
next
case (*next-d f*)
then show *?case* **using** *nlength-upd* **by** *simp*
(*metis One-nat-def ndropn-nlength ndropn-upd one-enat-def*)
next
case (*chop-d f1 f2*)
then show *?case*
proof −
have *c1*: $(\sigma \models\ \ f1\ schop\ f2\ ) =$
$(\exists i.\ enat\ i \leq nlength\ \sigma \wedge (ntaken\ i\ \sigma \models f1) \wedge (ndropn\ i\ \sigma \models f2))$ **by** *simp*
have *c2*: $(upd\ \sigma\ n\ l \models\ \ f1\ schop\ f2\ ) =$
$(\exists i.\ enat\ i \leq nlength\ (upd\ \sigma\ n\ l) \wedge (ntaken\ i\ (upd\ \sigma\ n\ l) \models f1) \wedge$
$(ndropn\ i\ (upd\ \sigma\ n\ l) \models f2))$
by *simp*
have *c3*: $(\exists i.\ enat\ i \leq nlength\ \sigma \wedge (ntaken\ i\ \sigma \models f1) \wedge (ndropn\ i\ \sigma \models f2)) \Longrightarrow$

```
        (∃ i. enat i ≤ nlength (upd σ n l) ∧ (ntaken i (upd σ n l) ⊨ f1) ∧
             (ndropn i (upd σ n l) ⊨ f2))
  proof −
   assume a0: (∃ i. enat i ≤ nlength σ ∧ (ntaken i σ ⊨ f1) ∧ (ndropn i σ ⊨ f2))
   show (∃ i. enat i ≤ nlength (upd σ n l) ∧ (ntaken i (upd σ n l) ⊨ f1) ∧
             (ndropn i (upd σ n l) ⊨ f2))
   proof −
   obtain i where c31: enat i ≤ nlength σ ∧ (ntaken i σ ⊨ f1) ∧ (ndropn i σ ⊨ f2)
     using a0 by blast
   have c32: n ∉ fvars f1
     using chop-d.prems(1) by auto
   have c33: nlength (ntaken i l) = nlength (ntaken i σ)
     by (simp add: chop-d.prems(2))
   have c34: (upd (ntaken i σ) n (ntaken i l)  ⊨ f1)
       using chop-d.IH(1)[of (ntaken i l) (ntaken i σ) ]  c31 c32 c33 by blast
   have c35: n ∉ fvars f2
     using chop-d.prems(1) by auto
   have c36: nlength (ndropn i l) = nlength (ndropn i σ)
     by (simp add: chop-d.prems(2))
   have c37: (upd (ndropn i σ) n (ndropn i l) ⊨ f2)
    using chop-d.IH(2)[of (ndropn i l) (ndropn i σ) ] c31 c35 c36 by blast
   have c38: (upd (ntaken i σ) n (ntaken i l)) = ntaken i (upd σ n l)
     by (simp add: c31 chop-d.prems(2) ntaken-upd)
   have c39: (upd (ndropn i σ) n (ndropn i l)) = ndropn i (upd σ n l)
     by (simp add: c31 chop-d.prems(2) ndropn-upd)
   have c40:  enat i ≤ nlength (upd σ n l)
   by (simp add: c31 chop-d.prems(2) nlength-upd)
   have c41: (ntaken i (upd σ n l) ⊨ f1)
   using c34 c38 by auto
   have c42: (ndropn i (upd σ n l) ⊨ f2)
   using c37 c39 by auto
   show ?thesis
   using c40 c41 c42 by auto
  qed
 qed
have c5: (∃ i. enat i ≤ nlength (upd σ n l) ∧ (ntaken i (upd σ n l) ⊨ f1) ∧
             (ndropn i (upd σ n l) ⊨ f2)) ⟹
        (∃ i. enat i ≤ nlength σ ∧ (ntaken i σ ⊨ f1) ∧ (ndropn i σ ⊨ f2))
proof −
 assume a1: (∃ i. enat i ≤ nlength (upd σ n l) ∧ (ntaken i (upd σ n l) ⊨ f1) ∧
             (ndropn i (upd σ n l) ⊨ f2))
 show (∃ i. enat i ≤ nlength σ ∧ (ntaken i σ ⊨ f1) ∧ (ndropn i σ ⊨ f2))
 proof −
  obtain i where c51: enat i ≤ nlength (upd σ n l) ∧ (ntaken i (upd σ n l) ⊨ f1) ∧
             (ndropn i (upd σ n l) ⊨ f2)
  using a1 by blast
  have c52: nlength (upd σ n l) = nlength σ
  by (simp add: chop-d.prems(2) nlength-upd)
  have c53: (ntaken i (upd σ n l)) = upd (ntaken i σ) n (ntaken i l)
  using c51 c52 chop-d.prems(2) ntaken-upd by auto
```

have *c54*: $n \notin fvars\ f1$
using *chop-d.prems(1)* **by** *auto*
have *c55*: $(ntaken\ i\ \sigma \models f1)$
by (*metis c51 c53 c54 chop-d.IH(1) chop-d.prems(2) ntaken-nlength*)
have *c56*: $n \notin fvars\ f2$
using *chop-d.prems(1)* **by** *auto*
have *c57*: $(ndropn\ i\ (upd\ \sigma\ n\ l)) = upd\ (ndropn\ i\ \sigma)\ n\ (ndropn\ i\ l)$
using *c51 c52 chop-d.prems(2) ndropn-upd* **by** *auto*
have *c58*: $(ndropn\ i\ \sigma \models f2)$
by (*metis c51 c56 c57 chop-d.IH(2) chop-d.prems(2) ndropn-nlength*)
show *?thesis*
using *c51 c52 c55 c58* **by** *auto*
qed
qed
show *?thesis*
using *c1 c2 c3 c5* **by** *blast*
qed
next
case (*schopstar-d f*)
then show *?case*
proof −
have *s1*: $(\sigma \models schopstar\ f\ ) = (\exists\, l1.\ nnth\ l1\ 0 = 0 \wedge$
$nfinite\ l1 \wedge$
$nidx\ l1 \wedge$
$enat\ (nlast\ l1) = nlength\ \sigma \wedge$
$nfinite\ \sigma \wedge$
$(\forall\, i.\ enat\ i < nlength\ l1 \longrightarrow$
$(nsubn\ \sigma\ (nnth\ l1\ i)$
$(nnth\ l1\ (Suc\ i)) \models f)))$ **by** *simp*
have *s2*: $((upd\ \sigma\ n\ l) \models schopstar\ f\ ) = (\exists\, l1.\ nnth\ l1\ 0 = 0 \wedge$
$nfinite\ l1 \wedge$
$nidx\ l1 \wedge$
$enat\ (nlast\ l1) = nlength\ (upd\ \sigma\ n\ l) \wedge$
$nfinite\ (upd\ \sigma\ n\ l) \wedge$
$(\forall\, i.\ enat\ i < nlength\ l1 \longrightarrow$
$(nsubn\ (upd\ \sigma\ n\ l)\ (nnth\ l1\ i)$
$(nnth\ l1\ (Suc\ i)) \models f)))$ **by** *simp*
have *s3*: $(\exists\, l1.\ nnth\ l1\ 0 = 0 \wedge$
$nfinite\ l1 \wedge$
$nidx\ l1 \wedge$
$enat\ (nlast\ l1) = nlength\ \sigma \wedge$
$nfinite\ \sigma \wedge$
$(\forall\, i.\ enat\ i < nlength\ l1 \longrightarrow$
$(nsubn\ \sigma\ (nnth\ l1\ i)$
$(nnth\ l1\ (Suc\ i)) \models f))) \Longrightarrow$
$(\exists\, l1.\ nnth\ l1\ 0 = 0 \wedge$
$nfinite\ l1 \wedge$
$nidx\ l1 \wedge$
$enat\ (nlast\ l1) = nlength\ (upd\ \sigma\ n\ l) \wedge$
$nfinite\ (upd\ \sigma\ n\ l) \wedge$

```
    (∀ i. enat i < nlength l1 ⟶
        (nsubn (upd σ n l) (nnth l1 i)
          (nnth l1 (Suc i)) ⊨ f)))
proof −
 assume a2: (∃ l1. nnth l1 0 = 0 ∧
    nfinite l1 ∧
    nidx l1 ∧
    enat (nlast l1) = nlength σ ∧
    nfinite σ ∧
    (∀ i. enat i < nlength l1 ⟶
        (nsubn σ (nnth l1 i)
          (nnth l1 (Suc i)) ⊨ f)))
 show (∃ l1. nnth l1 0 = 0 ∧
    nfinite l1 ∧
    nidx l1 ∧
    enat (nlast l1) = nlength (upd σ n l) ∧
    nfinite (upd σ n l) ∧
    (∀ i. enat i < nlength l1 ⟶
        (nsubn (upd σ n l) (nnth l1 i)
          (nnth l1 (Suc i)) ⊨ f)))
 proof −
  obtain l1 where s31: nnth l1 0 = 0 ∧
    nfinite l1 ∧
    nidx l1 ∧
    enat (nlast l1) = nlength σ ∧
    nfinite σ ∧
    (∀ i. enat i < nlength l1 ⟶
        (nsubn σ (nnth l1 i)
          (nnth l1 (Suc i)) ⊨ f))
   using a2 by blast
  have s32: nlength (upd σ n l) = nlength σ
  by (simp add: nlength-upd schopstar-d.prems(2))
  have s33: ⋀i. enat i < nlength l1 ⟶
            nlength (nsubn l (nnth l1 i) (nnth l1 (Suc i))) =
            nlength(nsubn σ (nnth l1 i) (nnth l1 (Suc i)))
            by (simp add: nsubn-nlength schopstar-d.prems(2))
  have s33: ⋀i.   enat i < nlength l1 ⟶
            (nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f) =
             (upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
                 (nsubn l (nnth l1 i) (nnth l1 (Suc i))) ⊨ f)
  using fvars.simps(6) s33 schopstar-d.IH schopstar-d.prems(1) by blast
  have s34: ⋀i. enat i < nlength l1 ⟶
            (upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
                 (nsubn l (nnth l1 i) (nnth l1 (Suc i)))) =
            (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)))
  by (metis eSuc-enat enat-ord-simps(1) ileI1 nidx-all-le-nlast nidx-expand nsubn-upd
      order-less-imp-le s31 schopstar-d.prems(2))
  have s35: (∀ i. enat i < nlength l1 ⟶
            (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
  using s31 s33 s34 by auto
```

```
  have s36: nfinite (upd σ n l)
  by (metis nlength-eq-enat-nfiniteD s31 s32)
  show ?thesis
  using s31 s32 s35 s36 by auto
 qed
qed
have s4: (∃ l1. nth l1 0 = 0 ∧
      nfinite l1 ∧
      nidx l1 ∧
      enat (nlast l1) = nlength (upd σ n l) ∧
      nfinite (upd σ n l) ∧
      (∀ i. enat i < nlength l1 ⟶
          (nsubn (upd σ n l) (nth l1 i)
            (nth l1 (Suc i)) ⊨ f))) ⟹
       (∃ l1. nth l1 0 = 0 ∧
      nfinite l1 ∧
      nidx l1 ∧
      enat (nlast l1) = nlength σ ∧
      nfinite σ ∧
      (∀ i. enat i < nlength l1 ⟶
          (nsubn σ (nth l1 i)
            (nth l1 (Suc i)) ⊨ f)))
 proof −
  assume a3: (∃ l1. nth l1 0 = 0 ∧
      nfinite l1 ∧
      nidx l1 ∧
      enat (nlast l1) = nlength (upd σ n l) ∧
      nfinite (upd σ n l) ∧
      (∀ i. enat i < nlength l1 ⟶
          (nsubn (upd σ n l) (nth l1 i)
            (nth l1 (Suc i)) ⊨ f)))
  show (∃ l1. nth l1 0 = 0 ∧
      nfinite l1 ∧
      nidx l1 ∧
      enat (nlast l1) = nlength σ ∧
      nfinite σ ∧
      (∀ i. enat i < nlength l1 ⟶
          (nsubn σ (nth l1 i)
            (nth l1 (Suc i)) ⊨ f)))
  proof −
   obtain l1 where s41: nth l1 0 = 0 ∧
      nfinite l1 ∧
      nidx l1 ∧
      enat (nlast l1) = nlength (upd σ n l) ∧
      nfinite (upd σ n l) ∧
      (∀ i. enat i < nlength l1 ⟶
          (nsubn (upd σ n l) (nth l1 i)
            (nth l1 (Suc i)) ⊨ f))
    using a3 by blast
  have s42: nlength (upd σ n l) = nlength σ
```

by (*simp add*: *nlength-upd schopstar-d.prems*(*2*))
have *s43*: $\bigwedge i.\ \ enat\ i < nlength\ l1 \longrightarrow$
$nlength\ (nsubn\ l\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i))) =$
$nlength(nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)))$
by (*simp add*: *nsubn-nlength schopstar-d.prems*(*2*))
have *s44*: $\bigwedge i.\ \ enat\ i < nlength\ l1 \longrightarrow$
$(nsubn\ (upd\ \sigma\ n\ l)\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i))) =$
$upd\ (nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)))\ n$
$(nsubn\ l\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)))$
by (*metis eSuc-enat enat-ord-simps*(*1*) *ileI1 nidx-all-le-nlast nidx-expand nsubn-upd*
order-less-imp-le s41 s42 schopstar-d.prems(*2*))
have *s45*: $\bigwedge i.\ enat\ i < nlength\ l1 \longrightarrow$
$(upd\ (nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)))\ n$
$(nsubn\ l\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i))) \models f) =$
$(nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)) \models f)$
using *fvars.simps*(*6*) *s43 schopstar-d.IH schopstar-d.prems*(*1*) **by** *blast*
have *s46*: $(\forall i.\ enat\ i < nlength\ l1 \longrightarrow$
$(nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)) \models f))$
using *s41 s44 s45* **by** *auto*
have *s47*: *nfinite* σ
by (*metis nlength-eq-enat-nfiniteD s41 s42*)
show *?thesis*
using *s41 s42 s46 s47* **by** *auto*
qed
qed
show *?thesis*
using *s4 s1 s2 s3* **by** *blast*
qed
next
case (*omega-d f*)
then show *?case*
proof −
have *o1*: $(\sigma \models omega\ f\ ) = (\exists l1.\ \neg\ nfinite\ l1 \wedge$
$nnth\ l1\ 0 = 0 \wedge$
$nidx\ l1 \wedge$
$(\forall i.\ enat\ (nnth\ l1\ i) \leq nlength\ \sigma) \wedge$
$(\forall i.\ nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)) \models f))$
by *simp*
have *o2*: $((upd\ \sigma\ n\ l) \models omega\ f\ ) =\ \ (\exists l1.\ \neg nfinite\ l1 \wedge$
$nnth\ l1\ 0 = 0 \wedge$
$nidx\ l1 \wedge$
$(\forall i.\ enat\ (nnth\ l1\ i) \leq nlength\ (upd\ \sigma\ n\ l)) \wedge$
$(\forall i.\ (nsubn\ (upd\ \sigma\ n\ l)\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)) \models f)))$
by *simp*
have *o3*: $(\exists l1.\ \neg\ nfinite\ l1 \wedge$
$nnth\ l1\ 0 = 0 \wedge$
$nidx\ l1 \wedge$
$(\forall i.\ enat\ (nnth\ l1\ i) \leq nlength\ \sigma) \wedge$
$(\forall i.\ nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)) \models f)) \Longrightarrow$
$(\exists l1.\ \neg nfinite\ l1 \wedge$

nnth l1 0 = 0 ∧
nidx l1 ∧
(∀ i. enat (nnth l1 i) ≤ nlength (upd σ n l)) ∧
(∀ i. (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)) ⊨ f)))
proof −
assume *a5*: *(∃ l1. ¬ nfinite l1 ∧*
nnth l1 0 = 0 ∧
nidx l1 ∧
(∀ i. enat (nnth l1 i) ≤ nlength σ) ∧
(∀ i. nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
show *(∃ l1. ¬nfinite l1 ∧*
nnth l1 0 = 0 ∧
nidx l1 ∧
(∀ i. enat (nnth l1 i) ≤ nlength (upd σ n l)) ∧
(∀ i. (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)) ⊨ f)))
proof −
obtain *l1* **where** *o31*: *¬ nfinite l1 ∧*
nnth l1 0 = 0 ∧
nidx l1 ∧
(∀ i. enat (nnth l1 i) ≤ nlength σ) ∧
(∀ i. nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f)
using *a5* **by** *blast*
have *o32*: *(∀ i. enat (nnth l1 i) ≤ nlength (upd σ n l))*
by (*simp add*: *nlength-upd o31 omega-d.prems(2)*)
have *o33*: *⋀i.*
nlength (nsubn l (nnth l1 i) (nnth l1 (Suc i))) =
nlength(nsubn σ (nnth l1 i) (nnth l1 (Suc i)))
by (*simp add*: *nsubn-nlength omega-d.prems(2)*)
have *o33*: *⋀i.*
(nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f) =
(upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
(nsubn l (nnth l1 i) (nnth l1 (Suc i))) ⊨ f)
using *fvars.simps(7) omega-d.IH omega-d.prems(1) o33* **by** *blast*
have *o34*: *⋀i.*
(upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
(nsubn l (nnth l1 i) (nnth l1 (Suc i)))) =
(nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)))
by (*metis Suc-n-not-le-n enat-ord-simps(1) linorder-le-cases nfinite-ntaken nidx-less-eq*
nsubn-upd ntaken-all o31 omega-d.prems(2))
have *o35*: *(∀ i.*
(nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
using *o31 o33 o34* **by** *auto*
show *?thesis*
using *o31 o32 o35* **by** *blast*
qed
qed
have *o4*: *(∃ l1. ¬nfinite l1 ∧*
nnth l1 0 = 0 ∧
nidx l1 ∧
(∀ i. enat (nnth l1 i) ≤ nlength (upd σ n l)) ∧

```
      (∀ i. (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))) ⟹
    (∃ l1. ¬ nfinite l1 ∧
      nnth l1 0 = 0 ∧
      nidx l1 ∧
      (∀ i. enat (nnth l1 i) ≤ nlength σ) ∧
      (∀ i. nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
 proof −
  assume a6: (∃ l1. ¬nfinite l1 ∧
      nnth l1 0 = 0 ∧
      nidx l1 ∧
      (∀ i. enat (nnth l1 i) ≤ nlength (upd σ n l)) ∧
      (∀ i. (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)) ⊨ f)))
  show (∃ l1. ¬ nfinite l1 ∧
      nnth l1 0 = 0 ∧
      nidx l1 ∧
      (∀ i. enat (nnth l1 i) ≤ nlength σ) ∧
      (∀ i. nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
  proof −
   obtain l1 where o41: ¬nfinite l1 ∧
      nnth l1 0 = 0 ∧
      nidx l1 ∧
      (∀ i. enat (nnth l1 i) ≤ nlength (upd σ n l)) ∧
      (∀ i. (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
    using a6 by blast
  have o42: (∀ i. enat (nnth l1 i) ≤ nlength σ)
  using nlength-upd o41 omega-d.prems(2) by auto
  have o43: ⋀i.
            nlength (nsubn l (nnth l1 i) (nnth l1 (Suc i))) =
              nlength(nsubn σ (nnth l1 i) (nnth l1 (Suc i)))
              by (simp add: nsubn-nlength omega-d.prems(2))
  have o44: ⋀i.
            (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i))) =
            upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
                (nsubn l (nnth l1 i) (nnth l1 (Suc i)))
    by (metis enat-ord-simps(1) less-Suc-eq less-le-not-le linorder-not-less nfinite-ntaken
        nidx-less-eq nsubn-upd ntaken-all o41 o42 omega-d.prems(2))
  have o45: ⋀i.
              (upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
                   (nsubn l (nnth l1 i) (nnth l1 (Suc i))) ⊨ f) =
              (nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f)
  using fvars.simps(7) o43 omega-d.IH omega-d.prems(1) by blast
  have o46: (∀ i.
           (nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
  using o41 o44 o45 by auto
  show ?thesis
  using o41 o42 o46 by blast
 qed
qed
show ?thesis
using o1 o2 o3 o4 by blast
```

qed
next
case (*exists-d x1 f*)
then show *?case*
proof −
have *e1*: (*σ* ⊨ *Ex x1. f*) = (∃ *l. nlength σ* = *nlength l* ∧ ((*upd σ x1 l*) ⊨ *f*))
by *simp*
have *e2*: (*upd σ n l* ⊨ *Ex x1. f*) =
(∃ *l1. nlength* (*upd σ n l*) = *nlength l1* ∧ ((*upd* (*upd σ n l*) *x1 l1*) ⊨ *f*))
by *simp*
have *e3*: (∃ *l. nlength σ* = *nlength l* ∧ ((*upd σ x1 l*) ⊨ *f*)) ⟹
(∃ *l1. nlength* (*upd σ n l*) = *nlength l1* ∧ ((*upd* (*upd σ n l*) *x1 l1*) ⊨ *f*))
by (*metis Diff-iff empty-iff exists-d.IH exists-d.prems*(*1*) *exists-d.prems*(*2*)
fvars.simps(*8*) *insert-iff nlength-upd upd-absorb upd-swap*)
have *e4*: (∃ *l1. nlength* (*upd σ n l*) = *nlength l1* ∧ ((*upd* (*upd σ n l*) *x1 l1*) ⊨ *f*)) ⟹
(∃ *l. nlength σ* = *nlength l* ∧ ((*upd σ x1 l*) ⊨ *f*))
by (*metis Diff-iff empty-iff exists-d.IH exists-d.prems*(*1*) *exists-d.prems*(*2*) *fvars.simps*(*8*)
insert-iff nlength-upd upd-absorb upd-swap)
show *?thesis*
using *e1 e2 e3 e4* **by** *blast*
qed
qed

lemma *inot-defs* [*simp*]:
(*σ* ⊨ *inot f*) = *Not* (*σ* ⊨ *f*)
by (*simp add*: *inot-d-def*)

lemma *ior-defs* [*simp*]:
(*σ* ⊨ *f1 ior f2*) = ((*σ* ⊨ *f1*) ∨ (*σ* ⊨ *f2*))
by (*metis inot-defs ior-d-def semantics-qpitl.simps*(*3*))

lemma *iand-defs* [*simp*]:
(*σ* ⊨ *f1 iand f2*) = ((*σ* ⊨ *f1*) ∧ (*σ* ⊨ *f2*))
by (*simp add*: *iand-d-def*)

lemma *ieqv-defs* [*simp*]:
(*σ* ⊨ *f1 ieqv f2*) = ((*σ* ⊨ *f1*) = (*σ* ⊨ *f2*))
by (*metis iand-defs ieqv-d-def semantics-qpitl.simps*(*3*))

lemma *itrue-defs* [*simp*]:
(*σ* ⊨ *itrue*)
by (*simp add*: *itrue-d-def*)

lemma *empty-defs* [*simp*]:
(*σ* ⊨ *empty*) = ((*nlength σ*) = *0*)
by (*auto simp add*: *empty-d-def more-d-def*)
(*metis co.enat.exhaust-sel enat-le-plus-same*(*2*) *plus-1-eSuc*(*2*))

lemma *forall-defs* [*simp*]:

$(\sigma \models Fa\ v.\ f) = (\forall\, l.\ nlength\ l = nlength\ \sigma \longrightarrow (upd\ \sigma\ v\ l \models f))$
using *forall-d-def* **by** *auto*

lemma *sometimes-defs*:
$(\sigma \models diamond\ f) = (\exists\, i.\ i \leq nlength\ \sigma \wedge ((ndropn\ i\ \sigma) \models f))$
by (*auto simp add*: *sometimes-d-def*)

lemma *always-defs*:
$(\sigma \models box\ f) = (\forall\, i.\ i \leq nlength\ \sigma \longrightarrow ((ndropn\ i\ \sigma \models f)))$
by (*auto simp add*: *always-d-def sometimes-defs*)

lemma *fin-defs*:
$(\sigma \models (fin\ f)) \longleftrightarrow nfinite\ \sigma \wedge (\ (ndropn\ (the\text{-}enat(nlength\ \sigma))\ \sigma) \models f)$
apply (*auto simp add*: *sometimes-defs fin-d-def*)
apply (*metis ndropn-nlength nfinite-ndropn nlength-eq-enat-nfiniteD zero-enat-def*)
apply (*metis enat-diff-cancel-left idiff-self not-enat-eq order-refl the-enat.simps*)
by (*metis enat-the-enat idiff-self infinity-ileE is-NNil-def is-NNil-ndropn ndropn-nlast*)

lemma *ifinite-defs*:
$(\sigma \models ifinite) \longleftrightarrow nfinite\ \sigma$
apply (*auto simp add*: *finite-d-def*)
using *nfinite-conv-nlength-enat* **apply** *fastforce*
by (*metis enat-ord-simps*(*3*) *enat-the-enat ndropn-nlast ndropn-nlength nlength-NNil order-refl*)

lemma *inf-defs*:
$(\sigma \models inf) \longleftrightarrow \neg\ nfinite\ \sigma$
by (*auto simp add*: *inf-d-def ifinite-defs*)

lemma *chop-defs*:
$(\sigma \models f;g) \longleftrightarrow (\exists\, i.\ i \leq nlength\ \sigma \wedge ((ntaken\ i\ \sigma) \models f) \wedge ((ndropn\ i\ \sigma) \models g)) \vee (\neg nfinite\ \sigma \wedge (\sigma \models f))$
by (*auto simp add*: *chop-d-def inf-defs*)

lemma *di-defs*:
$(\sigma \models di\ f) \longleftrightarrow (\exists\, i.\ i \leq nlength\ \sigma \wedge ((ntaken\ i\ \sigma) \models f)) \vee (\neg\ nfinite\ \sigma \wedge (\sigma \models f)\ )$
by (*auto simp add*: *di-d-def chop-defs*)

lemma *bi-defs*:
$(\sigma \models bi\ f) \longleftrightarrow (\forall\, i.\ i \leq nlength\ \sigma \longrightarrow ((ntaken\ i\ \sigma) \models f)) \wedge (\neg nfinite\ \sigma \longrightarrow (\sigma \models f))$
by (*auto simp add*: *di-defs bi-d-def*)

lemma *df-defs*:
$(\sigma \models df\ f) \longleftrightarrow (\exists\, i.\ i \leq nlength\ \sigma \wedge ((ntaken\ i\ \sigma) \models f))$
by (*auto simp add*: *df-d-def*)

lemma *bf-defs*:
$(\sigma \models bf\ f) \longleftrightarrow (\forall\, i.\ i \leq nlength\ \sigma \longrightarrow ((ntaken\ i\ \sigma) \models f))$
by (*auto simp add*: *df-defs bf-d-def*)

lemma *da-defs* :

$(\sigma \models da\ f) =$
$(\ (\exists\ n\ k\ .\ \ (\ n \leq k \wedge k \leq nlength\ \sigma \wedge (\ (nsubn\ \sigma\ n\ k) \models f))) \vee$
$(\neg nfinite\ \sigma \wedge (\exists\, n.\ (\ (ndropn\ n\ \sigma) \models f)))$
$)$
apply (*auto simp add*: *da-d-def chop-defs*)
apply (*metis enat-min-eq le-add2 ntaken-ndropn plus-enat-simps*(*1*))
apply (*metis enat-le-plus-same*(*2*) *enat-min-eq enat-ord-simps*(*1*) *ntaken-ndropn plus-enat-simps*(*1*))
apply (*metis enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat nsubn-def1 order-subst2*)
using *is-NNil-imp-nfinite is-NNil-ndropn nfinite-ndropn nle-le* **by** *blast*

lemma *ba-defs*:
$(\sigma \models ba\ f) \longleftrightarrow (\ (\forall\ n\ k.\ \ (\ n \leq k \wedge k \leq nlength\ \sigma \longrightarrow (\ (nsubn\ \sigma\ n\ k) \models f)) \wedge (\neg nfinite\ \sigma \longrightarrow (\ (ndropn\ n\ \sigma) \models f)))$
$)$
by (*auto simp add*: *ba-d-def da-defs*)

lemma *sda-defs* :
$(\sigma \models sda\ f) =$
$(\ (\exists\ n\ k\ .\ \ (\ n \leq k \wedge k \leq nlength\ \sigma \wedge (\ (nsubn\ \sigma\ n\ k) \models f)))$
$)$
apply (*auto simp add*: *sda-d-def chop-defs*)
apply (*metis add-diff-cancel-right' enat-min-eq le-add2 nsubn-def1 plus-enat-simps*(*1*))
by (*metis enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat nsubn-def1 order.trans*)

lemma *sba-defs*:
$(\sigma \models sba\ f) \longleftrightarrow (\ (\forall\ n\ k.\ \ (\ n \leq k \wedge k \leq nlength\ \sigma \longrightarrow (\ (nsubn\ \sigma\ n\ k) \models f))\ )$
$)$
by (*auto simp add*: *sba-d-def sda-defs*)

lemma *sfxbnd-defs*:
$(\sigma \models (sfxbnd\ v \simeq f1 : f2)) \longleftrightarrow$
$(\exists\, l.\ nlength\ \sigma = nlength\ l \wedge$
$(\forall\ i.\ i \leq nlength\ \sigma \longrightarrow (\ v\ |\in|\ (nnth\ (upd\ \sigma\ v\ l)\ i) \longleftrightarrow (upd\ (ndropn\ i\ \sigma)\ v\ (ndropn\ i\ l) \models f1))) \wedge$
$(upd\ \sigma\ v\ l \models f2))$
apply (*simp add*: *sfxbnd-d-def always-defs ndropn-nfirst*)
using *ndropn-upd nlength-upd* **by** *auto*

lemma *sfxbnd-defs-2*:
$(\sigma \models (sfxbnd\ v \simeq f1 : f2)) \longleftrightarrow$
$(\exists\, l.\ nlength\ \sigma = nlength\ l \wedge$
$(\forall\ i.\ i \leq nlength\ \sigma \longrightarrow (\ (nnth\ l\ i) \longleftrightarrow (upd\ (ndropn\ i\ \sigma)\ v\ (ndropn\ i\ l) \models f1))) \wedge$
$(upd\ \sigma\ v\ l \models f2))$
using *sfxbnd-defs nnth-upd*
by (*metis finsertCI fminus-iff*)

lemma *pfxbnd-defs*:
$(\sigma \models (pfxbnd\ v \simeq f1 : f2)) \longleftrightarrow$

$(\exists\, l.\ nlength\ \sigma = nlength\ l \wedge$
$(\forall\ i.\ i \leq nlength\ \sigma \longrightarrow$
$(\ v\ |\in|\ (nnth\ (upd\ \sigma\ v\ l)\ i) \longleftrightarrow$
$(upd\ (ntaken\ i\ \sigma)\ v\ (ntaken\ i\ l) \models f1))) \wedge$
$(upd\ \sigma\ v\ l \models f2))$
apply (*simp add*: *pfxbnd-d-def bf-defs ndropn-nfirst ntaken-nnth ntaken-nlast fin-defs*)
using *ntaken-upd nlength-upd* **by** *auto*

lemma *pfxbnd-defs-2*:
$(\sigma \models (pfxbnd\ v \simeq f1 : f2)) \longleftrightarrow$
$(\exists\, l.\ nlength\ \sigma = nlength\ l \wedge$
$(\forall\ i.\ i \leq nlength\ \sigma \longrightarrow (\ (nnth\ l\ i) \longleftrightarrow (upd\ (ntaken\ i\ \sigma)\ v\ (ntaken\ i\ l) \models f1))) \wedge$
$(upd\ \sigma\ v\ l \models f2))$
using *pfxbnd-defs nnth-upd*
by (*metis finsertCI fminus-iff*)

lemma *state-qpitl-defs*:
assumes *state-qpitl w*
shows $(\sigma \models w) \longleftrightarrow ((NNil\ (nfirst\ \sigma)) \models w)$
using *assms*
proof (*induct w arbitrary*: σ)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case*
by (*metis nfirst-nfuse nfuse-leftneutral nlast-NNil semantics-qpitl.simps*(*2*))
next
case (*iimp-d w1 w2*)
then show *?case* **by** *simp*
next
case (*next-d w*)
then show *?case* **by** *auto*
next
case (*chop-d w1 w2*)
then show *?case* **by** *auto*
next
case (*schopstar-d w*)
then show *?case* **by** *auto*
next
case (*omega-d w*)
then show *?case* **by** *auto*
next
case (*exists-d x1 w*)
then show *?case*
proof −
have *1*: $(\sigma \models Ex\ x1.\ w) \longleftrightarrow (\exists\, \sigma''.\ (\sigma'' \models w) \wedge similar\ \sigma\ x1\ \sigma'')$
using *exist-sigma-exist-value semantics-qpitl.simps*(*8*) **by** *presburger*

have *2*: $(NNil\ (nfirst\ \sigma) \models Ex\ x1.\ w) \longleftrightarrow (\exists\ \sigma''.\ (\sigma'' \models w) \wedge similar\ (NNil\ (nfirst\ \sigma))\ x1\ \sigma'')$
using *exist-sigma-exist-value semantics-qpitl.simps*(*8*) **by** *presburger*
have *3*: $(\exists \sigma''.\ (\sigma'' \models w) \wedge similar\ \sigma\ x1\ \sigma'') \Longrightarrow$
$(\exists\ \sigma''.\ (\sigma'' \models w) \wedge similar\ (NNil\ (nfirst\ \sigma))\ x1\ \sigma'')$
proof −
assume *a0*: $(\exists \sigma''.\ (\sigma'' \models w) \wedge similar\ \sigma\ x1\ \sigma'')$
show $(\exists\ \sigma''.\ (\sigma'' \models w) \wedge similar\ (NNil\ (nfirst\ \sigma))\ x1\ \sigma'')$
proof −
obtain σ'' **where** *4*: $(\sigma'' \models w) \wedge similar\ \sigma\ x1\ \sigma''$
using *a0* **by** *blast*
have *5*: *state-qpitl w*
using *exists-d.prems* **by** *auto*
have *6*: $(NNil\ (nfirst\ \sigma'') \models w)$
using *4 5 exists-d.hyps* **by** *blast*
have *7*: $similar\ (NNil\ (nfirst\ \sigma))\ x1\ (NNil\ (nfirst\ \sigma''))$
by (*metis* (*no-types*, *lifting*) *4 enat-defs*(*1*) *nfirst-eq-nnth-zero nlength-NNil nnth-NNil*
similar-defs-alt zero-order(*1*))
show *?thesis*
using *6 7* **by** *blast*
qed
qed
have *8*: $(\exists\ \sigma''.\ (\sigma'' \models w) \wedge similar\ (NNil\ (nfirst\ \sigma))\ x1\ \sigma'') \Longrightarrow$
$(\exists \sigma''.\ (\sigma'' \models w) \wedge similar\ \sigma\ x1\ \sigma'')$
proof −
assume *a1*: $(\exists\ \sigma''.\ (\sigma'' \models w) \wedge similar\ (NNil\ (nfirst\ \sigma))\ x1\ \sigma'')$
show $(\exists \sigma''.\ (\sigma'' \models w) \wedge similar\ \sigma\ x1\ \sigma'')$
proof −
obtain σ'' **where** *9*: $(\sigma'' \models w) \wedge similar\ (NNil\ (nfirst\ \sigma))\ x1\ \sigma''$
using *a1* **by** *blast*
have *10*: *state-qpitl w*
using *exists-d.prems* **by** *auto*
have *11*: $(NNil\ (nfirst\ \sigma'') \models w)$
using *9 10 exists-d.hyps* **by** *blast*
have *12*: $nlength\ \sigma = 0 \Longrightarrow similar\ \sigma\ x1\ \sigma''$
by (*metis 9 ndropn-0 ndropn-nlast nlength-eq-enat-nfiniteD ntake-0 ntake-NNil*
the-enat.simps zero-enat-def)
have *13*: $nlength\ \sigma > 0 \Longrightarrow similar\ \sigma\ x1\ (nappend\ (NNil\ (nfirst\ \sigma))\ (ndropn\ 1\ \sigma))$
by (*simp add*: *enat-defs*(*1*) *ndropn-Suc-conv-ndropn nfirst-eq-nnth-zero*)
have *14*: $nlength\ \sigma = 0 \Longrightarrow (\sigma'' \models w)$
by (*simp add*: *9*)
have *15*: $nlength\ \sigma > 0 \Longrightarrow (NNil\ (nfirst\ (nappend\ (NNil\ (nfirst\ \sigma))\ (ndropn\ 1\ \sigma)))) = (NNil\ (nfirst\ \sigma))$
by (*simp add*: *nfirst-eq-nnth-zero*)
have *16*: $nlength\ \sigma > 0 \Longrightarrow similar\ (NNil\ (nfirst\ \sigma))\ x1\ (NNil\ (nfirst\ \sigma''))$
by (*metis 9 le-numeral-extra*(*3*) *nlength-NNil ntaken-0 ntaken-all similar-defs-alt zero-enat-def*)
have *17*: $nlength\ \sigma > 0 \Longrightarrow \sigma = nappend\ (NNil\ (nfirst\ \sigma))\ (ndropn\ 1\ \sigma)$
by (*simp add*: *ndropn-Suc-conv-ndropn nfirst-eq-nnth-zero zero-enat-def*)
have *18*: $nlength\ \sigma > 0 \Longrightarrow similar\ (nappend\ (NNil\ (nfirst\ \sigma))\ (ndropn\ 1\ \sigma))\ x1\ (nappend\ (NNil\ (nfirst\ \sigma''))\ (ndropn\ 1\ \sigma))$
by *simp*

```
        (metis (no-types, lifting) 9 le-zero-eq nappend-NNil nfirst-eq-nnth-zero nlength-NNil nnth-0
         nnth-nappend similar-defs the-enat.simps)
          have 19: nlength σ > 0 ⟹ (nappend (NNil (nfirst σ'')) (ndropn 1 σ)) ⊨ w
         by (metis 10 9 exists-d.hyps nfirst-eq-nnth-zero nnth-nappend zero-enat-def zero-le)
       show ?thesis
       by (metis 12 17 18 19 9 i0-less)
     qed
   qed
   show ?thesis
   using 1 2 3 8 by blast
 qed
qed

lemma not-fvar-upd-state:
 assumes  n ∉ fvars w
          nlength l = nlength σ
          state-qpitl w
 shows  (NNil (nnth σ 0) ⊨ w) = (NNil (nnth (upd σ n l) 0) ⊨ w)
using assms by (metis  ndropn-0 ndropn-nfirst not-fvar-upd state-qpitl-defs)

lemma not-fvar-upd-nmap:
assumes n ∉ fvars w
        nlength l = nlength σ
        state-qpitl w
shows (nmap (λc. NNil c ⊨ w) (upd σ n l)) = (nmap (λc. NNil c ⊨ w) σ)
proof −
 have 1: nlength (nmap (λc. NNil c ⊨ w) (upd σ n l)) = nlength (nmap (λc. NNil c ⊨ w) σ)
 by (simp add: assms(2) nlength-upd)
 have 2: ⋀ i. enat i ≤ nlength (nmap (λc. NNil c ⊨ w) (upd σ n l)) ⟹
          (nnth (nmap (λc. NNil c ⊨ w) (upd σ n l)) i) =
          (nnth (nmap (λc. NNil c ⊨ w) σ) i)
   using assms nnth-nmap[of - (upd σ n l) (λc. NNil c ⊨ w)  ]
   by (metis (mono-tags, lifting) 1 ndropn-nfirst ndropn-nlength ndropn-upd nlength-nmap
       nnth-nmap not-fvar-upd state-qpitl-defs)
 show ?thesis
 using 1 2 nellist-eq-nnth-eq by blast
qed


lemma subst-suform:
 assumes n ∉ bvars f
         ⋀z . z ∈ fvars w ⟹ z ∉ bvars f
         state-qpitl w
 shows    (σ ⊨ suform f n w) = ((upd σ n (nmap (λ c. (NNil c) ⊨ w) σ)) ⊨ f)
using assms
proof (induction f arbitrary: σ w n)
case false-d
then show ?case by simp
next
case (atom-d x)
```

then show *?case*
proof −
have *a1*: $(\sigma \models (suform\ (\$x)\ n\ w)) = (if\ x = n\ then\ (\sigma \models w)\ else\ (x\ |\in|\ nfirst\ \sigma))$
by *simp*
have *a2*: $nlength\ \sigma = nlength\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)$
by *auto*
have *a3*: $(upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma) \models \$x) = (if\ x = n\ then\ (\sigma \models w)\ else\ (x\ |\in|\ nfirst\ \sigma))$
using *a2 nfirst-upd* **by** *simp*
(metis atom-d.prems(3) ndropn-0 ndropn-nfirst nnth-nmap state-qpitl-defs zero-enat-def zero-le)
show *?thesis*
using *a1 a3* **by** *blast*
qed
next
case *(iimp-d f1 f2)*
then show *?case* **by** *auto*
next
case *(next-d f)*
then show *?case* **by** *(auto simp add: nlength-upd ndropn-nmap ndropn-upd one-enat-def)*
next
case *(chop-d f1 f2)*
then show *?case*
proof −
have *c1*: $(\sigma \models suform\ (\ f1\ schop\ f2\ )\ n\ w) =$
$(\exists i.\ enat\ i \le nlength\ \sigma \wedge (ntaken\ i\ \sigma \models suform\ f1\ n\ w) \wedge (ndropn\ i\ \sigma \models suform\ f2\ n\ w))$
by *simp*
have *c2*: $(upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma) \models\ f1\ schop\ f2\ ) =$
$(\exists i.\ enat\ i \le nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1) \wedge$
$(ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2))$
by *simp*
have *c3*: $(\exists i.\ enat\ i \le nlength\ \sigma \wedge (ntaken\ i\ \sigma \models suform\ f1\ n\ w) \wedge (ndropn\ i\ \sigma \models suform\ f2\ n\ w))$
$\Longrightarrow$
$(\exists i.\ enat\ i \le nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1) \wedge$
$(ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2))$
proof −
assume *c4*: $(\exists i.\ enat\ i \le nlength\ \sigma \wedge (ntaken\ i\ \sigma \models suform\ f1\ n\ w) \wedge (ndropn\ i\ \sigma \models suform\ f2\ n\ w))$
show $(\exists i.\ enat\ i \le nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1) \wedge$
$(ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2))$
proof −
obtain *i* **where** *c5*: $enat\ i \le nlength\ \sigma \wedge (ntaken\ i\ \sigma \models suform\ f1\ n\ w) \wedge (ndropn\ i\ \sigma \models suform\ f2\ n\ w)$
using *c4* **by** *blast*
have *c6*: $nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) = nlength\ \sigma$
by *(simp add: nlength-upd)*
have *c7*: $n \notin bvars\ f1$
using *chop-d.prems(1)* **by** *auto*
have *c8*: $(ntaken\ i\ \sigma \models suform\ f1\ n\ w) = (ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1)$
using *chop-d.IH(1)[of n w (ntaken i σ)*]

using *c5 c7 chop-d.prems(2) chop-d.prems(3) ntaken-upd* **by** *auto*
have *c9*: $n \notin bvars\ f2$
using *chop-d.prems(1)* **by** *auto*
have *c10*: $(ndropn\ i\ \sigma \models suform\ f2\ n\ w) = (ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2)$
using *chop-d.IH(2)*[*of n w* (*ndropn i* σ)]
by (*metis Un-iff bvars.simps(5) c5 c9 chop-d.prems(2) chop-d.prems(3) ndropn-nmap ndropn-upd nlength-nmap*)
show *?thesis*
using *c10 c5 c6 c8* **by** *auto*
qed
qed
have *c11*: $(\exists i.\ enat\ i \leq nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1) \wedge$
$(ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2))$
$\Longrightarrow$
$(\exists i.\ enat\ i \leq nlength\ \sigma \wedge (ntaken\ i\ \sigma \models suform\ f1\ n\ w) \wedge (ndropn\ i\ \sigma \models suform\ f2\ n\ w))$
proof −
assume *c12*: $(\exists i.\ enat\ i \leq nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1) \wedge$
$(ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2))$
show $(\exists i.\ enat\ i \leq nlength\ \sigma \wedge (ntaken\ i\ \sigma \models suform\ f1\ n\ w) \wedge (ndropn\ i\ \sigma \models suform\ f2\ n\ w))$
proof −
obtain *i* **where** *c13*: $enat\ i \leq nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1) \wedge$
$(ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2)$
using *c12* **by** *blast*
have *c14*: $nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) = nlength\ \sigma$
by (*simp add*: *nlength-upd*)
have *c15*: $n \notin bvars\ f1$
using *chop-d.prems(1)* **by** *auto*
have *c16*: $(ntaken\ i\ \sigma \models suform\ f1\ n\ w) = (ntaken\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f1)$
using *chop-d.IH(1)*[*of n w* (*ntaken i* σ)]
using *c13 c14 c15 chop-d.prems(2) chop-d.prems(3) ntaken-upd* **by** *auto*
have *c17*: $n \notin bvars\ f2$
using *chop-d.prems(1)* **by** *auto*
have *c18*: $(ndropn\ i\ \sigma \models suform\ f2\ n\ w) = (ndropn\ i\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \models f2)$
using *chop-d.IH(2)*[*of n w* (*ndropn i* σ)]
by (*metis Un-iff bvars.simps(5) c13 c14 c17 chop-d.prems(2) chop-d.prems(3) ndropn-nmap ndropn-upd nlength-nmap*)
show *?thesis*
using *c13 c14 c16 c18* **by** *auto*
qed
qed
show *?thesis*
using *c1 c11 c2 c3* **by** *blast*
qed
next
case (*schopstar-d f*)
then show *?case*
proof −

have *s1*: $(\sigma \models suform\ (schopstar\ f\ )\ n\ w) =$
$(\exists\, l.\ nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ \sigma \wedge nfinite\ \sigma \wedge$
$(\forall\, i.\ enat\ i < nlength\ l \longrightarrow (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)))$
by *simp*
have *s2*: $(upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma) \models schopstar\ f\ ) =$
$(\exists\, l.\ nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$nfinite\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(\forall\, i.\ enat\ i < nlength\ l \longrightarrow (nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)))$
by *simp*
have *s3*: $(\exists\, l.\ nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ \sigma \wedge nfinite\ \sigma \wedge$
$(\forall\, i.\ enat\ i < nlength\ l \longrightarrow (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w))) \Longrightarrow$
$(\exists\, l.\ nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$nfinite\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(\forall\, i.\ enat\ i < nlength\ l \longrightarrow (nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)))$
proof −
assume *sa0*: $(\exists\, l.\ nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ \sigma \wedge nfinite\ \sigma \wedge$
$(\forall\, i.\ enat\ i < nlength\ l \longrightarrow (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)))$
show $(\exists\, l.\ nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$nfinite\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)) \wedge$
$(\forall\, i.\ enat\ i < nlength\ l \longrightarrow (nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)))$
proof −
obtain *l* **where** *s4*: $nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ \sigma \wedge nfinite\ \sigma \wedge$
$(\forall\, i.\ enat\ i < nlength\ l \longrightarrow (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w))$
using *sa0* **by** *blast*
have *s5*: $enat\ (nlast\ l) = nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))$
by (*simp add*: *nlength-upd s4*)

have *s6*: *nfinite* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ))
by (*metis nlength-eq-enat-nfiniteD s5*)
have *s7*: $\bigwedge$*i*. *enat i* < *nlength l* $\Longrightarrow$ (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *suform f n w*)
using *s4* **by** *blast*
have *s8*: $\bigwedge$*i*. *enat i* < *nlength l* $\Longrightarrow$
(*nsubn* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *f*)
proof −
fix *i*
assume *sa1*: *enat i* < *nlength l*
show (*nsubn* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *f*)
proof −
have *s9*: (*upd* (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))) $\models$ *f*)
using *schopstar-d.IH*[*of n w* (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))]
using *bvars.simps*(*6*) *s4 sa1 schopstar-d.prems*(*1*) *schopstar-d.prems*(*2*) *schopstar-d.prems*(*3*) **by** *blast*

have *s10*: (*nsubn* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) (*nnth l i*) (*nnth l* (*Suc i*))) =
upd (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) *n* (*nsubn* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ) (*nnth l i*) (*nnth l* (*Suc i*)))
using *nsubn-upd*[*of* σ (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ) (*nnth l i*) (*nnth l* (*Suc i*)) *n*]
by (*metis Suc-n-not-le-n eSuc-enat enat-ord-simps*(*1*) *ileI1 nidx-all-le-nlast nidx-less-eq nle-le nlength-nmap s4 sa1*)
have *s11*: (*nsubn* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ) (*nnth l i*) (*nnth l* (*Suc i*))) =
(*nmap* (λ*c*. *NNil c* $\models$ *w*) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))))
by (*simp add*: *ndropn-nmap nsubn-def1*)
have *s12*: (*upd* (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))))) =
(*nsubn* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) (*nnth l i*) (*nnth l* (*Suc i*)))
by (*simp add*: *s10 s11*)
show *?thesis*
using *s12 s9* **by** *auto*
qed
qed
show *?thesis*
using *s4 s5 s6 s8* **by** *blast*
qed
qed
have *s13*: (∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) ∧
nfinite (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *f*))) $\Longrightarrow$
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength* σ ∧ *nfinite* σ ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)) $\models$ *suform f n w*)))
proof −

assume *sa2*: (∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) ∧
nfinite (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*)))
show (∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*)))
proof −
obtain *l* **where** *s14*: *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) ∧
nfinite (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*))
using *sa2* **by** *blast*
have *s15*: *enat* (*nlast l*) = *nlength σ*
by (*simp add*: *nlength-upd s14*)
have *s16*: *nfinite σ*
by (*metis nlength-eq-enat-nfiniteD s15*)
have *s17*: ⋀*i*. *enat i* < *nlength l* ⟹
(*nsubn* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*)
using *s14* **by** *blast*
have *s18*: ⋀*i*. *enat i* < *nlength l* ⟹ (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*)
proof −
fix *i*
assume *sa3*: *enat i* < *nlength l*
show (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*)
proof −
have *s19*: (*nsubn* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*) (*nnth l i*) (*nnth l* (*Suc i*))) =
(*nmap* (λ*c*. *NNil c* ⊨ *w*) (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))
by (*simp add*: *ndropn-nmap nsubn-def1*)
have *s20*: (*nsubn* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*))) =
upd (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))) *n* (*nsubn* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*) (*nnth l i*) (*nnth l* (*Suc i*)))
using *nsubn-upd*[*of σ* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*) (*nnth l i*) (*nnth l* (*Suc i*)) *n*]
by (*metis Suc-n-not-le-n eSuc-enat enat-ord-simps*(*1*) *ileI1 linorder-le-cases nidx-all-le-nlast nidx-less-eq nlength-nmap s14 s15 sa3*)
have *s21*: (*upd* (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))) *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))) =
(*nsubn* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)))
using *s19 s20* **by** *presburger*
have *s22*: (*nsubn* (*upd σ n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*)
using *s14 sa3* **by** *blast*
have *s23*: (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*)

```
      using bvars.simps(6) s21 s22 schopstar-d.IH schopstar-d.prems(1) schopstar-d.prems(2)
       schopstar-d.prems(3) by presburger
      show ?thesis
      by (simp add: s23)
     qed
    qed
   show ?thesis
   using s14 s15 s16 s18 by blast
  qed
 qed
 show ?thesis
 using s1 s13 s2 s3 by blast
qed
next
case (omega-d f)
then show ?case
proof −
 have s1: (σ ⊨ suform (omega f ) n w) =
          (∃ l. ¬ nfinite l ∧
           nnth l 0 = 0 ∧
           nidx l ∧
           (∀ i. enat (nnth l i) ≤ nlength σ) ∧
           (∀ i. nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w))
  by simp
 have s2: (upd σ n (nmap (λc. NNil c ⊨ w) σ) ⊨ omega f ) =
          (∃ l. ¬ nfinite l ∧
           nnth l 0 = 0 ∧
           nidx l ∧
           (∀ i. enat (nnth l i) ≤ nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ))) ∧
           (∀ i. nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc i)) ⊨ f))
 by simp
 have s3: (∃ l. ¬ nfinite l ∧
           nnth l 0 = 0 ∧
           nidx l ∧
           (∀ i. enat (nnth l i) ≤ nlength σ) ∧
           (∀ i. nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w)) ⟹
          (∃ l. ¬ nfinite l ∧
           nnth l 0 = 0 ∧
           nidx l ∧
           (∀ i. enat (nnth l i) ≤ nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ))) ∧
           (∀ i. nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc i)) ⊨ f))
 proof −
  assume sa0: (∃ l. ¬ nfinite l ∧
           nnth l 0 = 0 ∧
           nidx l ∧
           (∀ i. enat (nnth l i) ≤ nlength σ) ∧
           (∀ i. nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w))
  show (∃ l. ¬ nfinite l ∧
           nnth l 0 = 0 ∧
           nidx l ∧
```

$(\forall\, i.\ enat\ (nnth\ l\ i) \le nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))) \wedge$
$(\forall\, i.\ nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f))$

proof −

obtain *l* **where** *s4*: $\neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge$
$(\forall\, i.\ enat\ (nnth\ l\ i) \le nlength\ \sigma) \wedge$
$(\forall\, i.\ nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)$
using *sa0* **by** *blast*

have *s5*: $(\forall\, i.\ enat\ (nnth\ l\ i) \le nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)))$
by (*simp add*: *nlength-upd s4*)

have *s7*: $\bigwedge i.\ \ enat\ i < nlength\ l \Longrightarrow (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)$
using *s4* **by** *blast*

have *s8*: $\bigwedge i.\ \ enat\ i < nlength\ l \Longrightarrow$
$(nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)$

proof −

fix *i*

assume *sa1*: *enat i < nlength l*

show $(nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)$

proof −

have *s9*: $(upd\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models f)$
using *omega-d.IH*[*of n w* $(nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))$]
using *omega-d.prems*(*1*) *omega-d.prems*(*2*) *omega-d.prems*(*3*) *s7 sa1* **by** *force*

have *s10*: $(nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) =$
$upd\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ n\ (nsubn\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))$
using *nsubn-upd*[*of* $\sigma\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\ n$]

by (*metis Suc-n-not-le-n eSuc-enat enat-ord-simps*(*1*) *ileI1 nidx-less-eq nle-le nlength-nmap s4 sa1*)

have *s11*: $(nsubn\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) =$
$(nmap\ (\lambda c.\ NNil\ c \models w)\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
by (*simp add*: *ndropn-nmap nsubn-def1*)

have *s12*: $(upd\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))) =$
$(nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))$
by (*simp add*: *s10 s11*)

show *?thesis*
using *s12 s9* **by** *auto*

qed

qed

show *?thesis*
by (*metis enat-ile linorder-not-less nlength-eq-enat-nfiniteD s4 s5 s8*)

qed

qed

have *s13*: $(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge$
$(\forall\, i.\ enat\ (nnth\ l\ i) \le nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))) \wedge$
$(\forall\, i.\ nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)) \Longrightarrow$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$

nnth l 0 = 0 ∧
nidx l ∧
(∀ *i*. *enat* (*nnth l i*) ≤ *nlength* *σ*) ∧
(∀ *i*. *nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*))
proof −
assume *sa2*: (∃ *l*. ¬ *nfinite l* ∧
nnth l 0 = 0 ∧
nidx l ∧
(∀ *i*. *enat* (*nnth l i*) ≤ *nlength* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*))) ∧
(∀ *i*. *nsubn* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*))
show (∃ *l*. ¬ *nfinite l* ∧
nnth l 0 = 0 ∧
nidx l ∧
(∀ *i*. *enat* (*nnth l i*) ≤ *nlength* *σ*) ∧
(∀ *i*. *nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*))
proof −
obtain *l* **where** *s14*: ¬ *nfinite l* ∧
nnth l 0 = 0 ∧
nidx l ∧
(∀ *i*. *enat* (*nnth l i*) ≤ *nlength* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*))) ∧
(∀ *i*. *nsubn* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*)
using *sa2* **by** *blast*
have *s15*: (∀ *i*. *enat* (*nnth l i*) ≤ *nlength* *σ*)
using *nlength-upd s14* **by** *auto*
have *s17*: ⋀*i*. (*nsubn* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*)
using *s14* **by** *blast*
have *s18*: ⋀*i*. (*nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*)
proof −
fix *i*
show (*nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*)
proof −
have *s19*: (*nsubn* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*) (*nnth l i*) (*nnth l* (*Suc i*))) =
(*nmap* (λ*c*. *NNil c* ⊨ *w*) (*nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*))))
by (*simp add*: *ndropn-nmap nsubn-def1*)
have *s20*: (*nsubn* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*))) =
upd (*nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*))) *n* (*nsubn* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*) (*nnth l i*) (*nnth l* (*Suc i*)))
using *nsubn-upd*[*of* *σ* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*) (*nnth l i*) (*nnth l* (*Suc i*)) *n*]
unfolding *nsubn-def1*
by (*metis Suc-ile-eq enat-ord-simps*(*1*) *le-SucI linorder-not-le nfinite-ntaken nidx-less-eq nlength-nmap ntaken-all s14 s15*)
have *s21*: (*upd* (*nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*))) *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) (*nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*))))) =
(*nsubn* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)))
using *s19 s20* **by** *presburger*
have *s22*: (*nsubn* (*upd* *σ* *n* (*nmap* (λ*c*. *NNil c* ⊨ *w*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *f*)
by (*simp add*: *s17*)
have *s23*: (*nsubn* *σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *suform f n w*)
using *bvars.simps*(*7*) *omega-d.IH omega-d.prems*(*1*) *omega-d.prems*(*2*) *omega-d.prems*(*3*) *s21 s22* **by** *presburger*

```
        show ?thesis
        by (simp add: s23)
       qed
      qed
     show ?thesis
     using s14 s15 s18 by blast
    qed
   qed
  show ?thesis
  using s1 s13 s2 s3 by blast
 qed
next
case (exists-d x1 f)
then show ?case
 proof −
  have e1: (σ ⊨ suform (Ex x1. f) n w) = (σ ⊨ Ex x1. suform f n w)
  using suform.simps(8) by presburger
  have e2: n = x1 ⟹ ?thesis
  using bvars.simps(8) exists-d.prems(1) by blast

  have e3: n ≠ x1 ⟹ ?thesis
   proof −
    assume ae0: n ≠ x1
    show ?thesis
    proof −
     have e4: n ∉ bvars f
     using exists-d.prems(1) by auto
     have e5: (upd σ n (nmap (λc. NNil c ⊨ w) σ) ⊨ Ex x1. f) =
              (∃ l. nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ)) = nlength l ∧
                   (upd (upd σ n (nmap (λc. NNil c ⊨ w) σ)) x1 l ⊨ f))
     by simp
     have e6: (⋀z. z ∈ fvars w ⟹ z ∉ bvars f)
     using exists-d.prems(2) by auto
     have e60:  x1 ∉ fvars w
     using exists-d.prems(2) by auto
     have e61: ⋀ σ l. nlength l = nlength σ ⟹ (σ ⊨ w) = ((upd σ x1 l) ⊨ w)
     using e60 not-fvar-upd by blast
     have e7: (σ ⊨ Ex x1. suform f n w) = (∃ l. nlength σ = nlength l ∧ (upd σ x1 l ⊨ suform f n w))
     by simp
     have e8: (∃ l. nlength σ = nlength l ∧ (upd σ x1 l ⊨ suform f n w)) ⟹
              (∃ l. nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ)) = nlength l ∧
                   (upd (upd σ n (nmap (λc. NNil c ⊨ w) σ)) x1 l ⊨ f))
     proof −
      assume ae1: (∃ l. nlength σ = nlength l ∧ (upd σ x1 l ⊨ suform f n w))
      show (∃ l. nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ)) = nlength l ∧
                   (upd (upd σ n (nmap (λc. NNil c ⊨ w) σ)) x1 l ⊨ f))
      proof −
       obtain l where e9:  nlength σ = nlength l ∧ (upd σ x1 l ⊨ suform f n w)
         using ae1 by blast
       have e10: (upd (upd σ x1 l) n (nmap (λc. NNil c ⊨ w) (upd σ x1 l)) ⊨ f)
```

using *e4 e6 e9 exists-d.IH exists-d.prems(3)* **by** *auto*
have *e11*: (*upd* (*upd* σ *x1 l*) *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*))) =
(*upd* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*))) *x1 l*)
by (*simp add*: *ae0 e9 nlength-upd upd-swap*)
have *e12*: *nlength* σ = *nlength* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*))
by (*simp add*: *e9 nlength-upd*)
have *e120*: *nlength* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) = *nlength* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)
by (*simp add*: *e12*)
have *e121*: $\bigwedge$ *i*. *i* $\leq$ *nlength* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) $\Longrightarrow$
(*nnth* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) *i*) =
(*nnth* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ) *i*)
by (*simp add*: *e60 e9 exists-d.prems(3) not-fvar-upd-nmap*)
have *e122*: (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) = (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)
using *e120 e121 nellist-eq-nnth-eq* **by** *blast*
show *?thesis*
using *e10 e11 e122 e9 nlength-upd* **by** *auto*
qed
qed
have *e13*: ($\exists$ *l*. *nlength* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) = *nlength l* $\wedge$
(*upd* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) *x1 l* $\models$ *f*)) $\Longrightarrow$
($\exists$ *l*. *nlength* σ = *nlength l* $\wedge$ (*upd* σ *x1 l* $\models$ *suform f n w*))
proof −
assume *ae1*: ($\exists$ *l*. *nlength* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) = *nlength l* $\wedge$
(*upd* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) *x1 l* $\models$ *f*))
show ($\exists$ *l*. *nlength* σ = *nlength l* $\wedge$ (*upd* σ *x1 l* $\models$ *suform f n w*))
proof −
obtain *l* **where** *e14*: *nlength* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) = *nlength l* $\wedge$
(*upd* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)) *x1 l* $\models$ *f*)
using *ae1* **by** *blast*
have *e15*: *nlength* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) = *nlength* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)
using *e14 nlength-upd* **by** *auto*
have *e16*: $\bigwedge$ *i*. *i* $\leq$ *nlength* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) $\Longrightarrow$
(*nnth* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) *i*) =
(*nnth* (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ) *i*)
using *e14 e60 exists-d.prems(3) nlength-upd not-fvar-upd-nmap* **by** *force*
have *e17*: (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) = (*nmap* (λ*c*. *NNil c* $\models$ *w*) σ)
using *e15 e16 nellist-eq-nnth-eq* **by** *blast*
have *e18*: (*upd* (*upd* σ *x1 l*) *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*))) =
(*upd* (*upd* σ *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*))) *x1 l*)
using *ae0 e14 nlength-upd upd-swap* **by** *auto*
have *e19*: (*upd* (*upd* σ *x1 l*) *n* (*nmap* (λ*c*. *NNil c* $\models$ *w*) (*upd* σ *x1 l*)) $\models$ *f*)
using *e14 e17 e18* **by** *presburger*
have *e20*: (*upd* σ *x1 l* $\models$ *suform f n w*)
by (*simp add*: *e19 e4 e6 exists-d.IH exists-d.prems(3)*)
show *?thesis*
using *e14 e20 nlength-upd* **by** *auto*
qed
qed
show *?thesis*
using *e1 e13 e5 e7 e8* **by** *blast*

```
  qed
 qed
 show ?thesis
 using e2 e3 by blast
 qed
qed

lemma rename-qpitl:
 assumes v1 ∉ bvars f
         v2 ∉ bvars f
         v1 ≠ v2
         v2 ∉ fvars f
 shows  (σ ⊨ Ex v1. f) = (σ ⊨ Ex v2. suform f v1 ($v2))
proof −
 have 1:  (σ ⊨ Ex v1. f) = (∃ l. nlength l = nlength σ ∧ ((upd σ v1 l) ⊨ f) )
 by force
 have 2:  (σ ⊨ Ex v2. suform f v1 ($v2)) =
          (∃ l. nlength l = nlength σ ∧ ((upd σ v2 l) ⊨ suform f v1 ($v2)))
 by force
 have 3:  (∃ l. nlength l = nlength σ ∧ ((upd σ v2 l) ⊨ suform f v1 ($v2))) ⟹
          (∃ l. nlength l = nlength σ ∧ ((upd σ v1 l) ⊨ f) )
  proof −
   assume a0: (∃ l. nlength l = nlength σ ∧ ((upd σ v2 l) ⊨ suform f v1 ($v2)))
   show (∃ l. nlength l = nlength σ ∧ ((upd σ v1 l) ⊨ f) )
   proof −
    obtain l where 4: nlength l = nlength σ ∧ ((upd σ v2 l) ⊨ suform f v1 ($v2))
    using a0 by blast
    have 5: ((upd σ v2 l) ⊨ suform f v1 ($v2)) =
            (upd (upd σ v2 l) v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)) ⊨ f)
     using  subst-suform[of v1 f $v2 (upd σ v2 l)]
     by (simp add: assms(1) assms(2))
    have 6: (upd (upd σ v2 l) v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) =
            (upd (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) v2 l)
    by (simp add: 4 assms(3) nlength-upd upd-swap)
    have 8: ((upd (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) v2 l) ⊨ f) =
             ((upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) ⊨ f)
    by (metis 4 assms(4) nlength-nmap nlength-upd not-fvar-upd)
    show ?thesis
    by (metis 4 5 6 8 nlength-nmap nlength-upd)
   qed
  qed
  have 9: (∃ l. nlength l = nlength σ ∧ ((upd σ v1 l) ⊨ f) ) ⟹
          (∃ l. nlength l = nlength σ ∧ ((upd σ v2 l) ⊨ suform f v1 ($v2)))
   proof −
    assume a1: (∃ l. nlength l = nlength σ ∧ ((upd σ v1 l) ⊨ f) )
    show (∃ l. nlength l = nlength σ ∧ ((upd σ v2 l) ⊨ suform f v1 ($v2)))
    proof −
     obtain l where 10: nlength l = nlength σ ∧ ((upd σ v1 l) ⊨ f)
     using a1 by blast
     have 11: nlength l = nlength (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)))
```

```
    by (simp add: 10 nlength-upd)
    have 12: v2 ∉ fvars f
    by (simp add: assms(4))
    have 13: (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)) ⊨ f) =
          ((upd (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) v2 l) ⊨ f)
    using not-fvar-upd[of v2 f l (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)))]
    11 12 by blast
    have 14: nlength (upd σ v1 l) =
          nlength (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)))
    by (simp add: 10 nlength-upd)
    have 15: ⋀ i . i ≤ nlength (upd σ v1 l) ⟹
          (nnth (upd σ v1 l) i) =
          (nnth (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) i)
     using nnth-upd[of σ (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)) - v1]
    by simp
     (metis (lifting) 10 11 14 finsert-iff fminus-iff nfirst-eq-nnth-zero nlength-nmap nnth-NNil nnth-upd
       upd-def)
    have 16: (upd σ v1 l) =
          (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)))
    by (simp add: 14 15 nellist-eq-nnth-eq)
    have 17: ((upd σ v2 l) ⊨ suform f v1 ($v2)) =
        (upd (upd σ v2 l) v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l)) ⊨ f)
    using assms(1) assms(2) subst-suform by fastforce
    have 18: (upd (upd σ v2 l) v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) =
        (upd (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) v2 l)
    by (simp add: 10 assms(3) nlength-upd upd-swap)
    have 19: ((upd (upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) v2 l) ⊨ f) =
          ((upd σ v1 (nmap (λc. NNil c ⊨ $v2) (upd σ v2 l))) ⊨ f)
    using 13 by auto
    show ?thesis
    using 10 13 16 17 18 by auto
   qed
  qed
 show ?thesis
 using 1 2 3 9 by blast
qed


lemma state-qpitl-list-defs:
 state-qpitl-list L ⟷ (∀ i<length L. state-qpitl (L!i))
proof (induct L)
case Nil
then show ?case unfolding state-qpitl-list-def by simp
next
case (Cons a L)
then show ?case
 proof −
  have 1: state-qpitl-list (a # L) ⟷ state-qpitl a ∧ state-qpitl-list L
    unfolding state-qpitl-list-def by simp
  have 2: (∀ i<length (a#L). state-qpitl ((a#L)!i)) ⟷
```

```
        state-qpitl ((a#L)!0) ∧
        (∀ i. 0<i ∧ i<length (a#L)⟶ state-qpitl ((a#L)!i))
      by (metis length-greater-0-conv list.discI zero-less-iff-neq-zero)
  have 3: state-qpitl ((a#L)!0) ⟷ state-qpitl a
    by simp
  have 4: (∀ i. 0<i ∧ i<length (a#L)⟶ state-qpitl ((a#L)!i)) ⟷
        (∀ i<length (L). state-qpitl (L!i))
   by auto
  show ?thesis
  using 1 2 3 4 local.Cons by presburger
 qed
qed



lemma big-ior-defs :
  (σ ⊨ (big-ior F)) ⟷ (∃ i< length F. (σ ⊨ F ! i))
proof (induct F)
case Nil
then show ?case unfolding big-ior-def by simp
next
case (Cons a F)
then show ?case
  proof −
   have 1: (σ ⊨ (big-ior (a#F))) ⟷ (σ ⊨ a ior (big-ior F))
      unfolding big-ior-def by simp
   have 2: (∃ i<length (a # F). σ ⊨ (a # F) ! i) ⟷
          (σ ⊨ (a # F) ! 0) ∨
          (∃ i. 0 < i ∧ i<length (a # F) ∧ (σ ⊨ (a # F) ! i))
     by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
   have 3: (σ ⊨ (a # F) ! 0) = (σ ⊨ a)
     by auto
   have 4: (∃ i. 0 < i ∧ i<length (a # F) ∧ (σ ⊨ (a # F) ! i)) ⟷
         (∃ i<length ( F). (σ ⊨ ( F) ! i))
     by auto
       (metis Suc-le-eq diff-Suc-Suc diff-zero gr0-conv-Suc less-Suc-eq-le)
    show ?thesis
    using 1 2 4 local.Cons by force
  qed
qed



lemma big-iand-defs :
  (σ ⊨ (big-iand F)) ⟷ (∀ i< length F. (σ ⊨ F ! i))
proof (induct F)
case Nil
then show ?case unfolding big-iand-def by simp
next
case (Cons a F)
then show ?case
```

```
  proof −
   have 1: (σ ⊨ (big-iand (a#F))) ⟷ (σ ⊨ a iand (big-iand F))
      unfolding big-iand-def by simp
   have 2: (∀ i<length (a # F). σ ⊨ (a # F) ! i) ⟷
            (σ ⊨ (a # F) ! 0) ∧
            (∀ i. 0 < i ∧ i<length (a # F) ⟶ (σ ⊨ (a # F) ! i))
     by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
   have 3: (σ ⊨ (a # F) ! 0) = (σ ⊨ a)
     by auto
   have 4: (∀ i. 0 < i ∧ i<length (a # F) ⟶ (σ ⊨ (a # F) ! i)) ⟷
          (∀ i<length ( F). (σ ⊨ ( F) ! i))
     by auto
    show ?thesis
    using 1 2 4 local.Cons by force
  qed
qed


lemma big-ior-map-iand-zip :
assumes length F =length G
shows (σ ⊨  big-ior (map2 iand-d  F G)) ⟷
      (∃ i < length F. (σ ⊨ F ! i iand G ! i))
using assms
proof (induct F arbitrary: G)
case Nil
then show ?case by (simp add: big-ior-defs)
next
case (Cons a F)
then show ?case
  proof (cases G=[])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-ior-defs)
  qed
qed

lemma big-iand-map-iimp-zip :
assumes length F =length G
shows (σ ⊨  big-iand (map2 iimp-d F G)) ⟷
      (∀ i < length F. (σ ⊨ F ! i iimp G ! i))
using assms
proof (induct F arbitrary: G)
case Nil
then show ?case by (simp add: big-iand-defs)
next
case (Cons a F)
then show ?case
  proof (cases G=[])
```

```
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-iand-defs)
  qed
qed


lemma map2-iand-defs :
 assumes length F = length G
 shows (∃ i<length G. σ ⊨ map2 iand-d F G ! i) ⟷
       (∃ i<length G. (σ ⊨  F !i) ∧ (σ ⊨ G ! i))
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed

lemma map2-iand-inot-defs :
 assumes length F = length G
 shows (∃ i<length G. σ ⊨ map2 (λx y. x iand inot y) F G ! i) ⟷
       (∃ i<length G. (σ ⊨ F! i) ∧ ¬(σ ⊨  G!i))
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed
```

5.2.2 Validity

definition *valid* :: *qpitl* ⇒ *bool* ((⊢ -) 5)
where (⊢ *f*) = (∀ σ. (σ ⊨ *f*))

lemma *itl-valid* [*simp*] :
(⊢ *f*) = (∀ σ. (σ ⊨ *f*))
by (*simp add*: *valid-def*)

lemma *itl-ieq*:
(⊢ *f ieqv g*) = (∀ σ. (σ ⊨ *f*) = (σ ⊨ *g*))
by (*auto simp add*: *ieqv-d-def iand-d-def ior-d-def inot-d-def*)

lemma *big-ior-iimp-subset-eq*:
assumes *set L1* ⊆ *set L2*
shows ⊢ *big-ior L1 iimp big-ior L2*
using *assms*
proof (*induct rule*: *list-induct2'*)
case *1*
then show *?case* **by** *simp*
next
case (*2 x xs*)
then show *?case* **by** *simp*
next
case (*3 y ys*)
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*4 x xs y ys*)
then show *?case*
proof (*cases x=y*)
case *True*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-ior-defs*)
(*metis diff-Suc-1 in-set-conv-nth length-Cons less-Suc-eq-0-disj list.simps*(*15*) *nth-Cons' subsetD*)
next
case *False*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-ior-defs*)
(*metis 4.prems in-set-conv-nth length-Cons subsetD*)
qed
qed

lemma *big-ior-ieqv-set-eq*:
assumes *set L1* = *set L2*
shows ⊢ *big-ior L1 ieqv big-ior L2*
using *assms*
by (*metis big-ior-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-ior-ieqv-remdups*:
shows ⊢ *big-ior L ieqv big-ior* (*remdups L*)
by (*metis big-ior-ieqv-set-eq set-remdups*)

lemma *big-iand-iimp-subset-eq*:
assumes *set L1* $\subseteq$ *set L2*
shows $\vdash$ *big-iand L2 iimp big-iand L1*
using *assms*
proof (*induct rule*: *list-induct2'*)
case *1*
then show *?case* **by** *simp*
next
case (*2 x xs*)
then show *?case* **by** (*simp add*: *big-iand-defs*)
next
case (*3 y ys*)
then show *?case* **by** *simp*
next
case (*4 x xs y ys*)
then show *?case*
proof (*cases x=y*)
case *True*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-iand-defs*)
(*metis* (*mono-tags*, *lifting*) *4.prems all-nth-imp-all-set length-Cons list-ball-nth subsetD*)
next
case *False*
then show *?thesis* **using** *4* **by** (*auto simp add*: *big-iand-defs*)
(*metis* (*no-types*, *opaque-lifting*) *4.prems all-nth-imp-all-set insert-absorb insert-subset length-Cons nth-mem*)
qed
qed

lemma *big-iand-ieqv-set-eq*:
assumes *set L1* = *set L2*
shows $\vdash$ *big-iand L1 ieqv big-iand L2*
using *assms*
by (*metis big-iand-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-iand-ieqv-remdups*:
$\vdash$ *big-iand L ieqv big-iand* (*remdups L*)
by (*metis big-iand-ieqv-set-eq set-remdups*)

lemma *big-ior-empty*:
($\vdash$ *big-ior* []) $\longleftrightarrow$ ($\vdash$ *ifalse*)
by (*auto simp add*: *big-ior-defs*)

lemma *big-ior-Cons*:
($\vdash$ *big-ior* (*f* # *L*)) $\longleftrightarrow$ ($\vdash$ *f ior* (*big-ior L*))
proof −
have *1*: ($\vdash$ *big-ior* (*f* # *L*)) $\Longrightarrow$ ($\vdash$ *f ior* (*big-ior L*))
by (*auto simp add*: *big-ior-defs*)
(*metis diff-Suc-1 gr0-conv-Suc less-Suc-eq less-imp-diff-less nth-Cons'*)
have *2*: ($\vdash$ *f ior* (*big-ior L*)) $\Longrightarrow$ ($\vdash$ *big-ior* (*f* # *L*))

by (*auto simp add*: *big-ior-defs*) *fastforce*
show *?thesis* **using** *1 2* **by** *auto*
qed

lemma *big-ior-Cons-alt*:
$\vdash$ *big-ior* (*f* # *L*) *ieqv f ior big-ior L*
unfolding *big-ior-def*
by *auto*

lemma *big-ior-1*:
assumes $\bigwedge$*i*. *i*< *length F* $\Longrightarrow$ ($\vdash$ *F* ! *i ieqv G* ! *i*)
length F = *length G*
shows $\vdash$ (*big-ior F*) *ieqv* (*big-ior G*)
using *assms*
by (*auto simp add*: *big-ior-defs*)

lemma *big-iand-1*:
assumes $\bigwedge$*i*. *i*< *length F* $\Longrightarrow$ ($\vdash$ *F* ! *i ieqv G* ! *i*)
length F = *length G*
shows $\vdash$ (*big-iand F*) *ieqv* (*big-iand G*)
using *assms*
by (*auto simp add*: *big-iand-defs*)

lemma *big-ior-big-iand-1*:
$\vdash$ (*inot* (*big-ior F*)) *ieqv* (*big-iand* (*map* (λ*x*. *inot x*) *F*))
by (*auto simp add*: *big-ior-defs big-iand-defs*)

lemma *big-ior-big-iand-2*:
$\vdash$ (*inot* (*big-iand F*)) *ieqv* (*big-ior* (*map* (λ*x*. *inot x*) *F*))
by (*auto simp add*: *big-ior-defs big-iand-defs*)

lemma *big-ior-append*:
($\vdash$ (*big-ior* (*L1*@*L2*)) *ieqv* (*big-ior L1*) *ior* (*big-ior L2*))
proof (*induct L1 arbitrary*: *L2*)
case *Nil*
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*Cons a L1a*)
then show *?case*
proof −
have *1*: $\vdash$ *big-ior* ((*a* # *L1a*) @ *L2*) *ieqv a ior* ((*big-ior* (*L1a* @ *L2*)))
using *big-ior-Cons-alt* **by** *auto*
have *2*: $\vdash$ *big-ior* (*L1a* @ *L2*) *ieqv big-ior L1a ior big-ior L2*
using *local.Cons* **by** *blast*
have *3*: $\vdash$ *a ior* ((*big-ior* (*L1a* @ *L2*))) *ieqv a ior* (*big-ior L1a ior big-ior L2*)

```
    using 1 2 by auto
   have 4: ⊢ big-ior (a # L1a) ieqv a ior (big-ior L1a)
    using big-ior-Cons-alt by blast
   have 5: ⊢  a ior (big-ior L1a ior big-ior L2) ieqv  big-ior (a # L1a) ior big-ior L2
       using 4 by auto
   show ?thesis using 1 3 5 by force
  qed
qed

lemma state-qpitl-list-append:
 shows   state-qpitl-list (L@L1) ⟷ state-qpitl-list L ∧ state-qpitl-list L1
proof (induct L arbitrary: L1)
case Nil
then show ?case
by (simp add: state-qpitl-list-defs)
next
case (Cons a L)
then show ?case
  proof −
   have 1: state-qpitl-list ((a # L) @ L1) ⟷ state-qpitl a ∧ state-qpitl-list (L @ L1)
     unfolding state-qpitl-list-def by auto
   have 2: state-qpitl-list (L @ L1) ⟷ state-qpitl-list L ∧ state-qpitl-list L1
     by (simp add: local.Cons)
   have 3: state-qpitl a ∧ state-qpitl-list L ∧ state-qpitl-list L1 ⟷
          (state-qpitl-list (a # L) ∧ state-qpitl-list L1)
     by (simp add: state-qpitl-list-def)
   show ?thesis using 1 3 local.Cons by auto
  qed
qed


lemma map2-map:
 assumes length L1 = length L2
 shows    map2 f (map g L1) (map h L2) = map2 (λx y. f (g x) (h y)) L1 L2
using assms
proof (induct L1 L2 rule:list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case by auto
qed


lemma map2-map-a:
assumes length L1 = length L2
shows (map2 (λx y. x iand next y) (map (λx1. x iand x1) L1) (map (λy1. y ior y1) L2)) =
      (map2 (λx1 y1. (x iand x1) iand next ( y ior y1)) L1 L2)
using assms map2-map by blast
```

lemma *map2-map-a-a*:
assumes *length L1 = length L2*
shows (*map2* (λ*x y*. *x iand next y*) (*map* (λ*x1*. *x iand x1*) *L1*) (*map* (λ*y1*. (*if y = ifalse then y1 else y ior y1*)) *L2*)) =
(*map2* (λ*x1 y1*. (*x iand x1*) *iand next* ((*if y = ifalse then y1 else y ior y1*))) *L1 L2*)
using *assms map2-map* **by** *blast*

lemma *map2-map-b*:
assumes *length L1 = length L2*
shows (*map2* (λ*x y*. *x iand next y*) (*map* (λ*x1*. (*inot x*) *iand x1*) *L1*) *L2*) =
(*map2* (λ*x1 y1*. ((*inot x*) *iand x1*) *iand next y1*) *L1 L2*)
using *assms map2-map*[*of L1 L2* (λ*x y*. *x iand next y*) (λ*x1*. (*inot x*) *iand x1*) *id*]
by *auto*

lemma *big-ior-big-ior-map2*:
assumes *length L1 = length L2*
⊢ *big-ior L1*
shows ⊢ *big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
using *assms*
proof (*induct L1 L2 rule*:*list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
by (*metis fst-def list.size*(*4*) *map-eq-conv map-fst-zip*)
qed

lemma *big-ior-split-off*:
assumes *length L1 = length L2*
(⊢ *big-ior L1*)
shows ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*y ior y1*))) *L1 L2*)) *ior*
(*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*x iand next y*) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*) *L1 L2*))
proof −
have *1*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*y ior y1*))) *L1 L2*)) *ieqv*
(*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand* (*next* (*y ior y1*))) *L1 L2*)))
by (*auto simp add*: *big-ior-defs*)
have *2*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. *x1 iand* (*next* (*y ior y1*))) *L1 L2*)) *ieqv*
((*next y iand* (*big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)))
ior
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))
by (*auto simp add*: *big-ior-defs*)
have *3*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*y ior y1*))) *L1 L2*)) *ieqv*
(*x iand* (*next y iand big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
ior

```
          (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 ))))
      by (auto simp add: big-ior-defs)
  have 4: ⊢ big-ior  (map2 (λx1 y1.  x1 ) L1 L2 )
       using assms big-ior-big-ior-map2 by blast
  have 5: ⊢ (x iand  (next y iand big-ior  (map2 (λx1 y1.  x1 )  L1 L2 )
            ior
            (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))) ieqv
           (x iand next y) ior (x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))
      using 4 by auto
  have 6: ⊢ (big-ior (map2 (λx1 y1. (inot x iand  x1) iand next  y1) L1 L2 )) ieqv
            (inot x iand (big-ior  (map2 (λx1 y1.  x1 iand ( next ( y1))) L1 L2 )))
    by (auto simp add: big-ior-defs)
  have 7: ⊢ (x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 ))) ior
            (inot x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 ))) ieqv
            (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 ))
      by (auto simp add: big-ior-defs)
  have 8 : ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand ( next ( y ior y1))) L1 L2 )) ior
            (big-ior (map2 (λx1 y1. (inot x iand x1) iand next  y1) L1 L2 )) ieqv
            (x iand next y) ior
            (x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 ))) ior
            (inot x iand (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))
        using 5 6 3 by auto
   show ?thesis using 7 8 by auto
qed

lemma big-ior-split-off-a:
assumes length L1 = length L2
      (⊢ big-ior L1)
shows ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 ))
ior
        (big-ior  (map2 (λx1 y1. (inot x iand  x1) iand next  y1)       L1 L2 )) ieqv
        (x iand next y) ior
        (big-ior  (map2 (λx y.  x iand next  y)  L1 L2 ))
proof −
 have 1: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2
)) ieqv
          (x iand (big-ior  (map2 (λx1 y1.  x1 iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )))
       by (auto simp add: big-ior-defs)
 have 2: ⊢ (big-ior  (map2 (λx1 y1.  x1 iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )) ieqv
           ((next y iand (big-ior  (map2 (λx1 y1.  x1 ) L1 L2 )))
           ior
           (big-ior  (map2 (λx1 y1.  x1 iand next ( y1)) L1 L2 )))
    proof (cases y=ifalse)
    case True
    then show ?thesis by simp
    next
    case False
    then show ?thesis by (auto simp add: big-ior-defs)
    qed
 have 3: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand ( next (if y = ifalse then y1 else y ior y1))) L1 L2 ))
```

ieqv
(*x iand* (*next y iand big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
ior
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))))
proof (*cases y=ifalse*)
case *True*
then show *?thesis* **by** (*auto simp add*: *big-ior-defs*)
next
case *False*
then show *?thesis* **by** (*auto simp add*: *big-ior-defs*)
qed
have *4*: ⊢ *big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
using *assms big-ior-big-ior-map2* **by** *blast*
have *5*: ⊢ (*x iand* (*next y iand big-ior* (*map2* (λ*x1 y1*. *x1*) *L1 L2*)
ior
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))) *ieqv*
(*x iand next y*) *ior* (*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))
using *4* **by** *auto*
have *6*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand* (*next* (*y1*))) *L1 L2*)))
by (*auto simp add*: *big-ior-defs*)
have *7*: ⊢ (*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ior*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))
by (*auto simp add*: *big-ior-defs*)
have *8* : ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand* (*next* (*if y = ifalse then y1 else y ior y1*))) *L1 L2*
)) *ior*
(*big-ior* (*map2* (λ*x1 y1*. (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*x iand next y*) *ior*
(*x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*))) *ior*
(*inot x iand* (*big-ior* (*map2* (λ*x1 y1*. *x1 iand next* (*y1*)) *L1 L2*)))
using *5 6 3* **by** *auto*
show *?thesis* **using** *7 8* **by** *auto*
qed

lemma *state-qpitl-list-map*:
assumes *state-qpitl f*
state-qpitl-list L
shows *state-qpitl-list* (*map* (λ *x*. *f iand x*) *L*)
using *assms*
proof (*induct L*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L*)
then show *?case*
proof −
have *1*: *state-qpitl-list* (*map* (λ*x*. *f iand x*) (*a* # *L*)) ⟷
state-qpitl (*f iand a*) ∧ *state-qpitl-list* ((*map* (λ*x*. *f iand x*) *L*))

```
      unfolding state-qpitl-list-def by auto
  have 2: state-qpitl-list (a # L) ⟷ state-qpitl a ∧ state-qpitl-list L
    unfolding state-qpitl-list-def by auto
  have 3:  state-qpitl ( f iand a)
    unfolding iand-d-def inot-d-def ior-d-def
    using 2 Cons.prems(2) assms(1) by auto
  show ?thesis
  using 1 2 3 Cons.hyps Cons.prems(2) assms(1) by blast
 qed
qed

lemma state-qpitl-list-map-alt:
 assumes state-qpitl f
       state-qpitl-list L
 shows  state-qpitl-list (map (λ x. x iand f) L)
using assms
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
then show ?case
  proof −
  have 1: state-qpitl-list (map (λx. x iand f ) (a # L)) ⟷
          state-qpitl ( a iand f ) ∧  state-qpitl-list ((map (λx. x iand f) L))
      unfolding state-qpitl-list-def by auto
  have 2: state-qpitl-list (a # L) ⟷ state-qpitl a ∧ state-qpitl-list L
    unfolding state-qpitl-list-def by auto
  have 3:  state-qpitl ( a iand f)
    unfolding iand-d-def inot-d-def ior-d-def
    using 2 Cons.prems(2) assms(1) by auto
  show ?thesis
  using 1 2 3 Cons.hyps Cons.prems(2) assms(1) by blast
 qed
qed

lemma map-fst-filter-sat:
assumes length L1 = length L2
shows  map fst (filter (λ (x,y). (∃ σ. σ ⊨ x))  (zip L1 L2)) =
       filter (λx. (∃ σ. σ ⊨ x)) L1
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis by simp
```

```
  next
  case False
  then show ?thesis
    using Cons by auto
  qed
qed


lemma big-ior-ieqv-filter-sat:
⊢ big-ior L ieqv big-ior ( filter (λx. (∃ σ. σ ⊨ x)) L)
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
then show ?case
using big-ior-Cons-alt by auto
qed


lemma total-filter-sat:
 assumes ⊢ big-ior L
 shows ⊢ big-ior ( filter (λx. (∃ σ. σ ⊨ x)) L)
using assms
using big-ior-ieqv-filter-sat by auto


lemma  big-ior-filter-sat-eqv:
assumes length L1 = length L2
shows ⊢ big-ior ( map (λ (x,y) . x iand (next  y))
                  (filter (λ (x,y). (∃ σ. σ ⊨ x))  (zip L1 L2)))
        ieqv
        big-ior (map2 (λ x y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis using Cons
    unfolding big-ior-def big-iand-def
      by simp
  next
  case False
  then show ?thesis
    proof −
     have 1: length xs = length ys
```

```
        by (simp add: Cons.hyps(1))
    have 2: ⊢ big-ior (map (λ(x, y). x iand next y)
                (filter (λ(x, y).  ∃σ. σ ⊨ x)
                 (zip xs ys)))
                ieqv big-ior (map2 (λx y. x iand next y) xs ys)
       using  Cons.hyps(2) False by fastforce
     show ?thesis
     using 2 big-ior-Cons-alt by auto
   qed
 qed
qed


lemma big-ior-rev-ieqv:
⊢ big-ior L ieqv big-ior (rev L)
by (auto simp add: big-ior-defs)
   (metis in-set-conv-nth length-rev set-rev,
    metis diff-less length-greater-0-conv less-nat-zero-code list.size(3) rev-nth zero-less-Suc)

lemma big-ior-map2-append:
assumes length L1= length N1
        length L2= length N2
shows  ⊢ (big-ior (map2 (λ x y . x iand (next y)) (L1@L2) (N1@N2))) ieqv
         (big-ior (map2 (λ x y . x iand (next y)) L1 N1)) ior
         (big-ior (map2 (λ x y . x iand (next y))  L2 N2))
using assms
proof (induct L1 N1 arbitrary: L2 N2 rule:list-induct2)
case Nil
then show ?case by (simp add: big-ior-defs)
next
case (Cons x xs y ys)
then show ?case
   by (metis Cons.hyps(1) big-ior-append length-Cons map-append zip-append)

qed

lemma map2-iand-next-defs :
 assumes length F = length G
 shows (∃ i<length G. σ ⊨ map2 (λx y. x iand (next y)) F G ! i) ⟷
       (∃ i<length G. (σ ⊨  F !i) ∧ (σ ⊨ (next (G ! i))))
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons.prems by auto
```

```
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed

lemma map2-iand-inot-next-defs :
 assumes length F = length G
         0<nlength σ
 shows (∃ i<length G. σ ⊨ map2 (λx y. x iand (next (inot y))) F G ! i) ⟷
        (∃ i<length G. (σ ⊨ F! i) ∧ ¬(σ ⊨  (next (G!i))))
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
   proof (cases F=[])
   case True
   then show ?thesis using Cons by simp
   next
   case False
   then show ?thesis
     proof −
      obtain b F1 where 1: F = b#F1
        using False by (meson neq-Nil-conv)
     have 2:  length F1 = length G
          using 1 Cons.prems by auto
     have 3: (∃ i<length G. σ ⊨ map2 (λx y. x iand next inot y) F1 G ! i) =
             (∃ i<length G. (σ ⊨ F1 ! i) ∧ ¬ (σ ⊨ next G ! i))
        using 2 Cons.hyps assms(2) by presburger
     have 4: (∃ i<length (a # G). σ ⊨ map2 (λx y. x iand next inot y) F (a # G) ! i) ⟷
             (σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! 0) ∨
             (∃ i. 0 < i ∧ i<length (a # G) ∧ ( σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! i))
         using 1  by (metis gr-zeroI length-greater-0-conv list.discI)
     have 5: (σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! 0) ⟷
             (σ ⊨ b iand next inot a)
       by auto
     have 6: (∃ i. 0 < i ∧ i<length (a # G) ∧ ( σ ⊨ map2 (λx y. x iand next inot y) (b#F1) (a # G) !
i)) ⟷
             (∃ i<length ( G).  ( σ ⊨ map2 (λx y. x iand next inot y) (F1) (G) ! i))
        by auto
          (metis Suc-less-eq Suc-pred)
     have 7: (∃ i<length (a # G). (σ ⊨ (b#F1) ! i) ∧ ¬ (σ ⊨ next (a # G) ! i))  ⟷
             ((σ ⊨ (b#F1) ! 0) ∧ ¬ (σ ⊨ next (a # G) ! 0)) ∨
             (∃ i. 0< i ∧ i<length (a # G) ∧ (σ ⊨ (b#F1) ! i) ∧ ¬ (σ ⊨ next (a # G) ! i))
        by (metis gr-zeroI length-greater-0-conv list.discI)
     have 8: ((σ ⊨ (b#F1) ! 0) ∧ ¬ (σ ⊨ next (a # G) ! 0)) ⟷
             (σ ⊨ b iand next inot a)
```

by *simp*
(*metis One-nat-def Suc-ile-eq assms(2) enat-defs(1) enat-defs(2)*)
have *9*: $(\exists\, i.\ 0< i \wedge i<length\ (a\ \#\ G) \wedge (\sigma \models (b\#F1)\ !\ i) \wedge \neg\ (\sigma \models next\ (a\ \#\ G)\ !\ i)) \longleftrightarrow$
$(\exists\, i<length\ (\ G).\ (\sigma \models (F1)\ !\ i) \wedge \neg\ (\sigma \models next\ (\ G)\ !\ i))$
by *auto*
(*metis Suc-diff-Suc Suc-less-eq diff-zero*, *metis Suc-less-eq Suc-pred*)
show *?thesis*
using *1 3 4 5 6 7 8 9* **by** *presburger*
qed
qed
qed

lemma *iand-big-ior-dist*:
assumes *length G-now = length G-next*
shows $\vdash$ *F-e iand big-ior (map2 (λ x y. x iand next y) G-now G-next) ieqv*
big-ior (map2 (λ x y. x iand next y)
(map (λx. x iand F-e) G-now) G-next)
using *assms* **by** (*auto simp add*: *big-ior-defs*)

lemma *big-ior-map2-defs*:
assumes *length L1 = length L2*
shows $(\sigma \models (big\text{-}ior\ (map2\ (\lambda x\ y.\ f\ x\ y)\ L1\ L2))) \longleftrightarrow$
$(\exists\, i<length\ L1.\ (\sigma \models f\ (L1!i)\ (L2!i)))$
using *assms*
proof (*induct L1 L2 rule*: *list-induct2*)
case *Nil*
then show *?case* **unfolding** *big-ior-def* **by** *auto*
next
case (*Cons x xs y ys*)
then show *?case* **using** *big-ior-def less-Suc-eq-0-disj* **by** *simp auto*
qed

lemma *big-ior-map2-schop-dis*:
assumes *length L1 = length L2*
shows $\vdash$ (*big-ior (map2 (λx y. f x y) L1 L2)) schop g ieqv*
(*big-ior (map2 (λx y. (f x y) schop g) L1 L2)*)
using *assms*
by (*auto simp add*: *big-ior-map2-defs*)

5.2.3 Quantifier lemmas

lemma *ExistSubst*:
assumes $n \notin bvars\ f$
$\bigwedge z\ .\ z \in fvars\ w \Longrightarrow z \notin bvars\ f$
state-qpitl w
shows $\vdash$ *suform f n w iimp (Ex n. f)*
using *assms subst-suform valid-def* **by** *force*

lemma *ForallSubst*:
assumes $n \notin$ *bvars f*
$\bigwedge z\ .\ z \in$ *fvars w* $\Longrightarrow z \notin$ *bvars f*
state-qpitl w
shows $\vdash$ (*Fa n. f*) *iimp suform f n w*
using *assms* **by** (*simp add*: *subst-suform*)

lemma *ExistsImp*:
assumes *v1* $\notin$ *fvars f1*
shows $\vdash$ (*Ex v1.* (*f1 iimp f2*)) *ieqv* (*f1 iimp* (*Ex v1. f2*))
proof −
have *1*: $\bigwedge \sigma$. ($\sigma \models$ (*Ex v1.* (*f1 iimp f2*))) = ($\sigma \models$ (*f1 iimp* (*Ex v1. f2*)))
by *simp*
(*metis assms nlength-nmap not-fvar-upd*)
show *?thesis*
using *1 itl-ieq* **by** *blast*
qed

lemma *ExistsSChopRight*:
assumes *v1* $\notin$ *fvars f2*
shows $\vdash$ (*Ex v1. f1 schop f2*) *iimp* (*Ex v1. f1*) *schop f2*
proof −
have *1*: $\bigwedge \sigma$. ($\sigma \models$ (*Ex v1. f1 schop f2*) *iimp* (*Ex v1. f1*) *schop f2*)
proof −
fix σ
show ($\sigma \models$ (*Ex v1. f1 schop f2*) *iimp* (*Ex v1. f1*) *schop f2*)
proof −
have *2*: ($\sigma \models$ (*Ex v1. f1 schop f2*)) =
($\exists\, l\ k.$ *nlength* σ = *nlength l* $\wedge$ *k*$\leq$ *nlength* σ $\wedge$
(*ntaken k* (*upd* σ *v1 l*) $\models$ *f1*) $\wedge$
(*ndropn k* (*upd* σ *v1 l*) $\models$ *f2*))
by *simp* (*metis nlength-upd*)
have *3*: ($\sigma \models$ (*Ex v1. f1*) *schop f2*) =
($\exists$ *k l.* $k \leq$ *nlength* σ $\wedge$ *nlength l* = *nlength* (*ntaken k* σ)
$\wedge$ (*upd* (*ntaken k* σ) *v1 l* $\models$ *f1*) $\wedge$ (*ndropn k* $\sigma \models$ *f2*))
by (*metis semantics-qpitl.simps*(*5*) *semantics-qpitl.simps*(*8*))
have *4*: ($\exists\, l\ k.$ *nlength* σ = *nlength l* $\wedge$ *k*$\leq$ *nlength* σ $\wedge$
(*ntaken k* (*upd* σ *v1 l*) $\models$ *f1*) $\wedge$
(*ndropn k* (*upd* σ *v1 l*) $\models$ *f2*)) $\Longrightarrow$
($\exists$ *k l.* $k \leq$ *nlength* σ $\wedge$ *nlength l* = *nlength* (*ntaken k* σ)
$\wedge$ (*upd* (*ntaken k* σ) *v1 l* $\models$ *f1*) $\wedge$ (*ndropn k* $\sigma \models$ *f2*))
by (*metis assms ndropn-nlength ndropn-upd not-fvar-upd ntaken-nlength ntaken-upd*)
show *?thesis*
using *2 3 4 semantics-qpitl.simps*(*3*) **by** *presburger*
qed
qed
show *?thesis*
using *1 valid-def* **by** *blast*

qed

lemma *ExistsSChopLeft*:
assumes *v1* $\notin$ *fvars f1*
shows $\vdash$ (*Ex v1. f1 schop f2*) *iimp f1 schop* (*Ex v1. f2*)
proof −
have *1*: $\bigwedge \sigma$. $\sigma \models$ (*Ex v1. f1 schop f2*) *iimp f1 schop* (*Ex v1. f2*)
proof −
fix σ
show $\sigma \models$ (*Ex v1. f1 schop f2*) *iimp f1 schop* (*Ex v1. f2*)
proof −
have *2*: ($\sigma \models$ (*Ex v1. f1 schop f2*)) =
($\exists$ *l k. nlength* σ = *nlength l* $\wedge$ *k*$\leq$ *nlength* σ $\wedge$
(*ntaken k* (*upd* σ *v1 l*) $\models$ *f1*) $\wedge$
(*ndropn k* (*upd* σ *v1 l*) $\models$ *f2*))
by *simp* (*metis nlength-upd*)
have *3*: ($\sigma \models$ *f1 schop* (*Ex v1. f2*)) =
($\exists$ *k l. k*$\leq$ *nlength* σ $\wedge$ *nlength l* = *nlength* (*ndropn k* σ) $\wedge$
(*ntaken k* $\sigma \models$ *f1*) $\wedge$ (*upd* (*ndropn k* σ) *v1 l* $\models$ *f2*))
by *force*
have *4*: ($\exists$ *l k. nlength* σ = *nlength l* $\wedge$ *k*$\leq$ *nlength* σ $\wedge$
(*ntaken k* (*upd* σ *v1 l*) $\models$ *f1*) $\wedge$
(*ndropn k* (*upd* σ *v1 l*) $\models$ *f2*)) $\Longrightarrow$
($\exists$ *k l. k*$\leq$ *nlength* σ $\wedge$ *nlength l* = *nlength* (*ndropn k* σ) $\wedge$
(*ntaken k* $\sigma \models$ *f1*) $\wedge$ (*upd* (*ndropn k* σ) *v1 l* $\models$ *f2*))
by (*metis assms ndropn-nlength ndropn-upd not-fvar-upd ntaken-nlength ntaken-upd*)
show *?thesis*
using *2 3 4 semantics-qpitl.simps*(*3*) **by** *presburger*
qed
qed
show *?thesis*
using *1 valid-def* **by** *blast*
qed

lemma *ForallGen*:
assumes $\vdash$ *f*
shows $\vdash$ *Fa v. f*
using *assms* **by** *simp*

lemma *ExistsI*:
assumes ($\sigma \models$ *f*)
shows ($\sigma \models$ *Ex v. f*)
using *assms exist-sigma-exist-value semantics-qpitl.simps*(*8*) *similar-refl*
by (*metis nlength-nmap upd-self*)

lemma *ExistsNoDep*:
assumes *v* $\notin$ *fvars f*
shows $\vdash$ (*Ex v . f*) *ieqv f*
using *ExistsI assms not-fvar-upd* **by** *auto*

lemma *ForaAllNoDep*:
assumes $v \notin fvars\ f$
shows $\vdash (Fa\ v.\ f)\ ieqv\ f$
using *assms*
using *nlength-nmap not-fvar-upd*
apply *simp*
using *nlength-nmap* **by** *blast*

lemma *ExistsEqvRule*:
assumes $\vdash f\ ieqv\ g$
shows $\vdash (Ex\ v.\ f)\ ieqv\ (Ex\ v.\ g)$
using *assms* **by** *auto*

lemma *ForallImpExists*:
$\vdash (Fa\ v.\ f)\ iimp\ (Ex\ v.\ f)$
by *simp*
(*metis exist-sigma-exist-value similar-refl*)

lemma *ExistsImpRuleDist*:
assumes $\vdash f\ iimp\ g$
shows $\vdash (Fa\ v.\ f)\ iimp\ (Ex\ v.\ g)$
using *assms* **by** *simp*
(*metis nlength-nmap*)

lemma *ExistsImpNoDepDist*:
assumes $v \notin fvars\ f$
$\vdash f\ iimp\ g$
shows $\vdash f\ iimp\ (Ex\ v.\ g)$
using *assms*
using *ExistsI* **by** *force*

lemma *ExistsOrDist-1*:
$\vdash (Ex\ v.\ h)\ iimp\ (Ex\ v.\ f\ ior\ h)$
by *auto*

lemma *ExistsOrDist-2*:
$\vdash (Ex\ v.\ h)\ iimp\ (Ex\ v.\ h\ ior\ f)$
by *auto*

lemma *ExistsOrDist-3*:
$\vdash (Ex\ v.\ f)\ ior\ (Ex\ v.\ h)\ iimp\ (Ex\ v.\ f\ ior\ h)$
by *auto*

lemma *ExistsOrDist*:
$\vdash (Ex\ v.\ f)\ ior\ (Ex\ v.\ h)\ ieqv\ (Ex\ v.\ f\ ior\ h)$
by *auto*

lemma *ExistsOrImport-1*:

assumes $v \notin fvars\ g$
shows $\vdash g\ iimp\ (Ex\ v.\ \ g\ ior\ f)$
using *assms*
using *ExistsI ior-defs semantics-qpitl.simps(3) valid-def* **by** *presburger*

lemma *ExistsOrImport-2*:
assumes $v \notin fvars\ g$
shows $\vdash (Ex\ v.\ f)\ iimp\ (Ex\ v.\ \ g\ ior\ f)$
using *assms*
using *ExistsOrDist-1* **by** *blast*

lemma *ExistsOrImport-3*:
assumes $v \notin fvars\ g$
shows $\vdash g\ ior\ (Ex\ v.\ f)\ iimp\ \ (Ex\ v.\ g\ ior\ \ \ f)$
using *assms*
using *ExistsI* **by** *fastforce*

lemma *ExistsOrImport-4*:
assumes $v \notin fvars\ g$
shows $\vdash\ \ (Ex\ v.\ g\ ior\ f)\ iimp\ \ g\ ior\ (Ex\ v.\ f)$
using *assms*
using *ExistsNoDep* **by** *auto*

lemma *ExistsOrImport*:
assumes $v \notin fvars\ g$
shows $\vdash\ \ g\ ior\ (Ex\ v.\ g\ ior\ f)\ ieqv\ (Ex\ v.\ g\ ior\ f)$
using *assms*
using *ExistsI* **by** *fastforce*

lemma *ExistsAndImport-1*:
assumes $v \notin fvars\ g$
shows $\vdash g\ iand\ (Ex\ v.\ f)\ iimp\ (Ex\ v.\ g\ iand\ f)$
using *assms*
using *not-fvar-upd* **by** *auto*

lemma *ExistsAndImport-2*:
assumes $v \notin fvars\ g$
shows $\vdash\ \ (Ex\ v.\ g\ iand\ f)\ iimp\ g\ iand\ (Ex\ v.\ \ \ f)$
using *assms*
using *not-fvar-upd* **by** *auto*

lemma *ExistsAndImport*:
assumes $v \notin fvars\ g$
shows $\vdash\ \ g\ iand\ (Ex\ v.\ \ \ f)\ ieqv\ (Ex\ v.\ g\ iand\ f)$
using *assms*
using *not-fvar-upd* **by** *auto*

lemma *ExistsAndDist*:
assumes $\vdash f\ iand\ g$
shows $\vdash (Ex\ v.\ f)\ iand\ (Ex\ v.\ g)$

using *assms*
using *ExistsI* **by** *fastforce*

lemma *ExistsAndNoDepDist*:
assumes $\vdash f$ *iand* g
$v \notin$ *fvars* f
shows $\vdash f$ *iand* (*Ex* v. g)
using *assms*
by (*meson ExistsI iand-defs valid-def*)

lemma *ForallE*:
assumes $\vdash$ *Fa* v. f
$\vdash f$ *iimp* g
shows $\vdash g$
using *assms*
by (*metis ExistsI forall-d-def inot-defs semantics-qpitl.simps*(*3*) *valid-def*)

lemma *ForallEqvRule*:
assumes $\vdash f$ *ieqv* g
shows $\vdash$ (*Fa* v. f) *ieqv* (*Fa* v. g)
using *assms* **by** *auto*

lemma *ForallAndDist*:
$\vdash$ (*Fa* v. f *iand* g) *ieqv* (*Fa* v. f) *iand* (*Fa* v. g)
by *auto*

lemma *ForallAndImport*:
assumes $v \notin$ *fvars* g
shows $\vdash g$ *iand* (*Fa* v. f) *ieqv* (*Fa* v. g *iand* f)
using *assms*
using *ForaAllNoDep ForallAndDist* **by** *force*

lemma *ForallOrImport*:
assumes $v \notin$ *fvars* g
shows $\vdash g$ *ior* (*Fa* v. f) *ieqv* (*Fa* v. g *ior* f)
using *assms*
using *not-fvar-upd* **by** *auto*

lemma *RightChopImpChop*:
assumes $\vdash f$ *iimp* g
shows $\vdash h$ *schop* f *iimp* h *schop* g
using *assms* **by** *auto*

lemma *ExistsImpChopRule*:
assumes $\vdash f$ *iimp* g
$v \notin$ *fvars* h
shows $\vdash$ (*Ex* v. (h *schop* f) *iimp* (h *schop* g))
using *assms*
by (*meson ExistsI RightChopImpChop valid-def*)

lemma *SChop-nfuse*:
shows $(\sigma \models f\ schop\ g) =$
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nlast\ \sigma 1 = nfirst\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$(\sigma 1 \models f) \wedge (\sigma 2 \models g))$
by *auto*
(*metis ndropn-nfirst nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast*,
metis linorder-le-cases ndropn-nfuse nfinite-conv-nlength-enat ntaken-all ntaken-nfuse the-enat.simps)

lemma *ExistsSChopRightNoDep*:
assumes $v \notin fvars\ g$
shows $\vdash (Ex\ v.\ f\ schop\ g)\ ieqv\ (Ex\ v.\ f)\ schop\ g$
proof −
have *1*: $\vdash (Ex\ v.\ f\ schop\ g)\ iimp\ (Ex\ v.\ f)\ schop\ g$
using *ExistsSChopRight assms* **by** *blast*
have *2*: $\vdash (Ex\ v.\ f)\ schop\ g\ iimp\ (Ex\ v.\ f\ schop\ g)$
proof −
have *3*: $\bigwedge \sigma.\ (\sigma \models (Ex\ v.\ f)\ schop\ g) \Longrightarrow (\sigma \models (Ex\ v.\ f\ schop\ g))$
proof −
fix σ
assume *a0*: $(\sigma \models (Ex\ v.\ f)\ schop\ g)$
show $(\sigma \models (Ex\ v.\ f\ schop\ g))$
proof −
have *4*: $(\exists\ k\ l.\ k \leq nlength\ \sigma \wedge nlength\ l = nlength(ntaken\ k\ \sigma)$
$\wedge ((upd\ (ntaken\ k\ \sigma)\ v\ l) \models f) \wedge (ndropn\ k\ \sigma \models g)\ )$
using *a0* **by** *fastforce*
obtain *k l* **where** *5*: $k \leq nlength\ \sigma \wedge nlength\ l = nlength(ntaken\ k\ \sigma)$
$\wedge ((upd\ (ntaken\ k\ \sigma)\ v\ l) \models f) \wedge (ndropn\ k\ \sigma \models g)$
using *4* **by** *blast*
have *6*: $\bigwedge l1.\ nlength\ l1 = nlength\ (ndropn\ k\ \sigma) \Longrightarrow$
$(\ (upd\ (ndropn\ k\ \sigma)\ v\ l1)\ \models g)$
using *5 assms not-fvar-upd* **by** *blast*
obtain *l1* **where** *7*: $nlength\ l1 = nlength\ (ndropn\ k\ \sigma) \wedge$
$nfirst\ l1 = nlast\ l \wedge$
$(\ (upd\ (ndropn\ k\ \sigma)\ v\ l1)\ \models g)$
using *6 nlength-nmap*
by (*metis ndropn-0 ndropn-nfirst nellist.exhaust nlength-NCons nlength-NNil nnth-0 nnth-NNil*)
have *70*: $enat\ k \leq nlength\ (nfuse\ l\ l1)$
by (*simp add*: *5 nfuse-nlength*)
have *8*: $ntaken\ k\ (nfuse\ l\ l1) = l$
by (*metis 5 enat-le-plus-same*(*2*) *gen-nlength-def min.absorb1 nfuse-def1 nlength-code*
ntaken-all ntaken-nappend1 ntaken-nlength)
have *9*: $ndropn\ k\ (nfuse\ l\ l1) = l1$
using *ndropn-nfuse*[*of l l1*]
by (*metis 7 70 8 min.absorb1 nfinite-ntaken ntaken-nlength the-enat.simps*)
have *10*: $(ntaken\ k\ (upd\ \sigma\ v\ (nfuse\ l\ l1)) \models f)$
by (*metis 5 7 8 nfuse-nlength nfuse-ntaken-ndropn ntaken-upd*)
have *11*: $(ndropn\ k\ (upd\ \sigma\ v\ (nfuse\ l\ l1)) \models g)$
by (*metis 5 7 9 ndropn-upd nfuse-nlength nfuse-ntaken-ndropn*)

```
      have 12: nlength σ = nlength (nfuse l l1)
      by (metis 5 7 nfuse-nlength nfuse-ntaken-ndropn-nlength)
      show ?thesis using 70 10 11 12
      using nlength-upd by fastforce
     qed
    qed
    show ?thesis
    using 3 semantics-qpitl.simps(3) valid-def by presburger
   qed
  show ?thesis
  using 1 2 ieqv-d-def by auto
qed

lemma ExistsSChopLeftNoDep:
 assumes v ∉ fvars g
 shows ⊢ (Ex v. g schop f) ieqv g schop (Ex v. f)
proof −
 have 1: ⊢ (Ex v. g schop f) iimp g schop (Ex v. f)
 using ExistsSChopLeft assms by blast
 have 2: ⊢  g schop (Ex v. f) iimp (Ex v. g schop f)
  proof −
   have 3: ⋀ σ. (σ ⊨  g schop (Ex v. f)) ⟹ (σ ⊨ (Ex v. g schop f))
    proof −
     fix σ
     assume a0: (σ ⊨  g schop (Ex v. f))
     show (σ ⊨ (Ex v. g schop f))
     proof −
      have 4: (∃ k l. k ≤ nlength σ ∧ nlength l = nlength (ndropn k σ) ∧
              (ntaken k σ ⊨ g) ∧ (upd (ndropn k σ) v l ⊨ f))
      using a0 by fastforce
      obtain k l where 5: k ≤ nlength σ ∧ nlength l = nlength (ndropn k σ) ∧
              (ntaken k σ ⊨ g) ∧ (upd (ndropn k σ) v l ⊨ f)
      using 4 by blast
      have 6: ⋀l1. nlength l1 = nlength (ntaken k σ)  ⟹
                 (upd (ntaken k σ) v l1 ⊨ g)
        using 5 assms nfinite-conv-nlength-enat not-fvar-upd by auto

      have 60: nlength (ntaken k σ) = k
      by (simp add: 5)
      have 61: nlength σ = 0 ⟹  ∃ l1. nlength l1 = nlength (ntaken k σ) ∧ (upd (ntaken k σ) v l1 ⊨ g) ∧
nfinite l1
              ∧ nlast l1 = nfirst l
       by (metis 5 6 le-zero-eq ndropn-0 nfirst-eq-nnth-zero nlength-eq-enat-nfiniteD nnth-nlast
           ntaken-all the-enat.simps the-enat-0)
      have 62: nlength σ >0 ∧ k> 0 ⟹ ∃ l1. nlength (nappend l1 (NNil (nnth l 0)))= nlength (ntaken k
σ) ∧
       (upd (ntaken k σ) v (nappend l1 (NNil (nnth l 0))) ⊨ g) ∧ nfinite l1
       using 6 60 5 by auto
       (metis One-nat-def Suc-pred add.right-neutral add-Suc-right idiff-enat-enat min.orderE
        ndropn-nlength nfinite-ntaken nle-le nlength-NNil nlength-nappend nlength-nmap ntaken-all
```

```
        ntaken-nlength one-enat-def plus-enat-simps(1))
    have 63: nlength σ > 0 ∧ k =0 ⟹ ∃ l1. nlength l1 = nlength (ntaken k σ) ∧ (upd (ntaken k σ) v
l1 ⊨ g) ∧ nfinite l1
            ∧ nlast l1 = nfirst l
    by (metis 6 nfinite-NNil nlast-NNil nlength-NNil ntaken-0)
    have 64: nlength σ = 0 ⟹ ?thesis
     by (metis 5 6 ndropn-0 nle-le nlength-upd ntaken-all semantics-qpitl.simps(5)
        semantics-qpitl.simps(8) the-enat.simps zero-enat-def zero-le)
    have 65: nlength σ > 0 ∧ k =0 ⟹ ?thesis
     by (metis 5 6 ndropn-0 nlength-upd ntaken-nlength ntaken-upd semantics-qpitl.simps(5)
        semantics-qpitl.simps(8) )
    have 66: nlength σ > 0 ∧ k >0 ⟹ ?thesis
     proof −
      assume a1: nlength σ > 0 ∧ k >0
      show ?thesis
      proof −
       have 70: ∃ l1. nlength (nappend l1 (NNil (nnth l 0)))= nlength (ntaken k σ) ∧
       (upd (ntaken k σ) v (nappend l1 (NNil (nnth l 0))) ⊨ g) ∧ nfinite l1
        using a1 62 by auto
       obtain l1 where 71: nlength (nappend l1 (NNil (nnth l 0)))= nlength (ntaken k σ) ∧
       (upd (ntaken k σ) v (nappend l1 (NNil (nnth l 0))) ⊨ g) ∧ nfinite l1
        using 70 by blast
       have 72: nlast (nappend l1 (NNil (nnth l 0))) = nfirst l
        by (simp add: 71 nfirst-eq-nnth-zero nlast-nappend)
       have 73: enat k ≤ nlength (nfuse (nappend l1 (NNil (nnth l 0))) l)
        by (metis 60 71 enat-le-plus-same(1) nfuse-nlength)
       have 74: ntaken k (nfuse (nappend l1 (NNil (nnth l 0))) l) = (nappend l1 (NNil (nnth l 0)))
        using 60 71 72 ntaken-nfuse[of (nappend l1 (NNil (nnth l 0))) l]
        nlength-eq-enat-nfiniteD by (metis the-enat.simps)
       have 75: ndropn k (nfuse (nappend l1 (NNil (nnth l 0))) l) = l
        by (metis 60 71 74 ndropn-nfuse nfinite-ntaken nfirst-eq-nnth-zero nlast-NNil
           nlast-nappend the-enat.simps)
       have 76: (ntaken k (upd σ v (nfuse (nappend l1 (NNil (nnth l 0))) l)) ⊨ g)
        by (metis 5 71 74 nfuse-nlength nfuse-ntaken-ndropn ntaken-upd)
       have 77: (ndropn k (upd σ v (nfuse (nappend l1 (NNil (nnth l 0))) l)) ⊨ f)
        by (metis 5 71 75 ndropn-upd nfuse-nlength nfuse-ntaken-ndropn)
       have 78: nlength σ = nlength (nfuse (nappend l1 (NNil (nnth l 0))) l)
        by (metis 5 71 nfuse-nlength nfuse-ntaken-ndropn-nlength)
       show ?thesis by simp
        (metis 5 76 77 78 nlength-upd)
      qed
     qed
    show ?thesis
    by (meson 64 65 66 i0-less less-nat-zero-code linorder-neqE-nat)
   qed
  qed
 show ?thesis
 using 3 semantics-qpitl.simps(3) valid-def by presburger
 qed
show ?thesis
```

using *1 2 ieqv-d-def* **by** *auto*
qed

lemma *ExistsExistsSChop*:
assumes $v \notin fvars\ g$
$y \notin fvars\ f$
shows $\vdash (Ex\ v.\ (Ex\ y.\ f\ schop\ g))\ ieqv\ (Ex\ v.\ f)\ schop\ (Ex\ y.\ g)$
proof −
have *1*: $\vdash (\ (Ex\ y.\ f\ schop\ g))\ ieqv\ (\ f\ schop\ (Ex\ y.\ \ g)\ )$
using *assms ExistsSChopLeftNoDep*[*of y f g*] **by** *blast*
have *2*: $\vdash (Ex\ v.\ (Ex\ y.\ f\ schop\ g))\ ieqv\ \ (Ex\ v.\ \ f\ schop\ (Ex\ y.\ \ g)\ )$
using *1 ExistsEqvRule* **by** *blast*
have *3*: $\vdash\ \ (Ex\ v.\ \ f\ schop\ (Ex\ y.\ \ g)\ )\ ieqv\ (Ex\ v.\ f)\ schop\ (Ex\ y.\ \ g)$
by (*metis Diff-iff ExistsSChopRightNoDep assms*(*1*) *fvars.simps*(*8*))
show *?thesis*
by (*meson 2 3 itl-ieq*)
qed

lemma *exists-elim-state-qpitl*:
assumes *state-qpitl w*
$n \notin bvars\ w$
shows $\vdash (Ex\ n.\ w)\ ieqv\ (suform\ w\ n\ itrue)\ ior\ (suform\ w\ n\ ifalse)$
proof −
have *1*: *state-qpitl itrue*
by (*simp add*: *inot-d-def itrue-d-def*)
have *2*: $(\bigwedge z.\ z \in fvars\ itrue \Longrightarrow z \notin bvars\ w)$
by (*simp add*: *inot-d-def itrue-d-def*)
have *3*: $\bigwedge \sigma.\ (\sigma \models (suform\ w\ n\ itrue)) = ((upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models itrue)\ \sigma)) \models w)$
using *subst-suform*[*of n w itrue*] *1 2 assms*(*2*) **by** *blast*
have *4*: *state-qpitl ifalse*
by *simp*
have *5*: $(\bigwedge z.\ z \in fvars\ ifalse \Longrightarrow z \notin bvars\ w)$
by *simp*
have *6*: $\bigwedge \sigma.\ (\sigma \models (suform\ w\ n\ ifalse)) = ((upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models ifalse)\ \sigma)) \models w)$
using *subst-suform*[*of n w ifalse*] *4 5 assms*(*2*) **by** *blast*
have *7*: $\bigwedge \sigma.\ (\sigma \models (Ex\ n.\ w)) = (\exists l.\ nlength\ \sigma = nlength\ l \wedge (upd\ \sigma\ n\ l \models w))$
by *simp*
have *8*: $\bigwedge \sigma.\ (\exists l.\ nlength\ \sigma = nlength\ l \wedge (upd\ \sigma\ n\ l \models w)) =$
$(\exists l.\ nlength\ \sigma = nlength\ l \wedge ((NNil\ (nfirst\ (upd\ \sigma\ n\ l)))\ \models w))$
using *assms*(*1*) *state-qpitl-defs* **by** *blast*
have *9*: $\bigwedge \sigma.\ ((upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models itrue)\ \sigma)) \models w) =$
$(NNil\ (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models itrue)\ \sigma))) \models w)$
using *assms*(*1*) *state-qpitl-defs* **by** *blast*
have *10*: $\bigwedge \sigma.\ \ (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models itrue)\ \sigma))) =$
$finsert\ n\ (nnth\ \sigma\ 0)$
by (*metis itrue-defs nfirst-eq-nnth-zero nfirst-upd nlength-nmap nnth-nmap zero-enat-def zero-le*)
have *11*: $\bigwedge \sigma.\ ((upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models ifalse)\ \sigma)) \models w) =$

$(NNil\ (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models ifalse)\ \sigma))) \models w)$
using *assms(1) state-qpitl-defs* **by** *blast*
have *12*: $\bigwedge \sigma.\ (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c.\ (NNil\ c) \models ifalse)\ \sigma))) =$
$(nnth\ \sigma\ 0)\ |-|\ \{|n|\}$
by (*metis i0-less nfirst-eq-nnth-zero nfirst-upd nle-le nlength-nmap nnth-nmap order-less-imp-le semantics-qpitl.simps(1) zero-enat-def*)
have *13*: $\bigwedge \sigma\ l.\ nlength\ \sigma = nlength\ l \Longrightarrow (nfirst\ (upd\ \sigma\ n\ l)) =$
$(if\ (nnth\ l\ 0)\ then\ finsert\ n\ (nnth\ \sigma\ 0)\ else\ (nnth\ \sigma\ 0)\ |-|\ \{|n|\})$
by (*metis ndropn-0 ndropn-nfirst nfirst-upd*)
have *14*: $\bigwedge \sigma.\ (\exists l.\ nlength\ \sigma = nlength\ l \wedge ((NNil\ (nfirst\ (upd\ \sigma\ n\ l)))\ \models w)) \longleftrightarrow$
$(NNil\ (finsert\ n\ (nnth\ \sigma\ 0)) \models w) \vee (NNil\ ((nnth\ \sigma\ 0)\ |-|\ \{|n|\}) \models w)$
by (*metis 10 12 ndropn-0 ndropn-nfirst nfirst-upd nlength-nmap*)
have *15*: $\bigwedge \sigma.\ (\sigma \models (Ex\ n.\ w)) \longleftrightarrow (\sigma \models (suform\ w\ n\ itrue)) \vee (\sigma \models (suform\ w\ n\ ifalse))$
by (*metis 10 12 14 3 6 7 assms(1) state-qpitl-defs*)
show *?thesis*
using *15 ior-defs itl-ieq* **by** *presburger*
qed

end

theory *QPITL-Deep-GNF* **imports**
QPITL-Deep

begin

This theory proves that all QPITL formulae can be written in Guarded Normal Form (GNF) [5, 2].

5.3 Guarded Normal Form

5.3.1 Definitions

definition *mutex*
where $mutex\ F =$
$(\forall\ k1\ k2.\ 0 \leq k1\ \wedge k1 < k2\ \wedge k2 < length\ F \longrightarrow (\vdash inot\ ((F!\ k1)\ iand\ (F!\ k2))))$

definition *mutex-single*
where $mutex\text{-}single\ f\ F =$
$(\forall\ k2.\ 0 \leq k2\ \wedge k2 < length\ F \longrightarrow (\vdash inot\ (f\ iand\ (F!\ k2))))$

definition *add-to-now-single*
where
$add\text{-}to\text{-}now\text{-}single\ f\ L = (\ (map\ (\lambda x.\ (\ f\ iand\ x)\ )\ L)@$
$(map\ (\lambda x.\ (\ inot\ f\ iand\ x))\ L))$

definition *add-to-next-single*
where
add-to-next-single f L = ((*map* (λ*x*. (*f ior x*)) *L*)@ *L*)

definition *add-to-single*
where
add-to-single f1 f2 L1 L2 =
filter (λ (*x*,*y*). (∃ σ. σ ⊨ *x*)) (*zip* (*add-to-now-single f1 L1*) (*add-to-next-single f2 L2*))

definition *add-to-next-single-a*
where
add-to-next-single-a f L = ((*map* (λ*x*. ((*if f* = *ifalse then x else f ior x*))) *L*)@ *L*)

definition *add-to-now*
where *add-to-now L1* = *foldr* (*add-to-now-single*) *L1* [*itrue*]

definition *add-to-next*
where *add-to-next L1* = *foldr* (*add-to-next-single*) *L1* [*ifalse*]

definition *add-to-next-a*
where *add-to-next-a L1* = *foldr* (*add-to-next-single-a*) *L1* [*ifalse*]

definition *add-to*
where *add-to L1 L2* = *filter* (λ (*x*,*y*). (∃σ . σ ⊨ *x*)) (*zip* (*add-to-now L1*) (*add-to-next L1*))

fun *GNF-empty* :: *qpitl* ⇒ *qpitl*
where
GNF-empty (*ifalse*) = *ifalse*
| *GNF-empty* ($*n*) = ($*n*)
| *GNF-empty* (*f iimp g*) = ((*inot* (*GNF-empty f*)) *ior* (*GNF-empty g*))
| *GNF-empty* (*next f*) = (*ifalse*)
| *GNF-empty* (*f schop g*) = ((*GNF-empty f*) *iand* (*GNF-empty g*))
| *GNF-empty* (*schopstar f*) = *itrue*
| *GNF-empty* (*omega f*) = *ifalse*
| *GNF-empty* (*Ex n. f*) = (*Ex n.* (*GNF-empty f*))

fun *GNF-state* :: *qpitl* ⇒ *qpitl list*
where
GNF-state (*ifalse*) = [*itrue*]
| *GNF-state* ($*n*) = [($*n*)]
| *GNF-state* (*f iimp g*) = (((*add-to-now* (*GNF-state f*))) @ (*GNF-state g*))
| *GNF-state* (*next f*) = [*itrue*]

| *GNF-state* (*f schop g*) = ((*GNF-state f*) @ (*map* (λ*x*. *x iand* (*GNF-empty f*)) (*GNF-state g*)))
| *GNF-state* (*schopstar f*) = (*GNF-state f*)
| *GNF-state* (*omega f*) = (*GNF-state f*)
| *GNF-state* (*Ex n. f*) = (*map* (λ*x*. (*Ex n. x*)) (*GNF-state f*))

fun *GNF-next* :: *qpitl* ⇒ *qpitl list*
where
GNF-next (*ifalse*) = [*ifalse*]
| *GNF-next* ($*n*) = [*itrue*]
| *GNF-next* (*f iimp g*) = ((*map* (λ*x*. (*inot x*)) (*add-to-next* (*GNF-next f*))) @ (*GNF-next g*))
| *GNF-next* (*next f*) = [*f*]
| *GNF-next* (*f schop g*) = ((*map* (λ*x*. *x schop g*) (*GNF-next f*)) @ (*GNF-next g*))
| *GNF-next* (*schopstar f*) = (*map* (λ*x*. *x schop* (*schopstar f*)) (*GNF-next f*))
| *GNF-next* (*omega f*) = (*map* (λ*x*. *x schop* (*omega f*)) (*GNF-next f*))
| *GNF-next* (*Ex n. f*) = (*map* (λ*x*. (*Ex n. x*)) (*GNF-next f*))

fun *GNF-next-a* :: *qpitl* ⇒ *qpitl list*
where
GNF-next-a (*ifalse*) = [*ifalse*]
| *GNF-next-a* ($*n*) = [*itrue*]
| *GNF-next-a* (*f iimp g*) = ((*map* (λ*x*. (*inot x*)) (*add-to-next-a* (*GNF-next-a f*))) @ (*GNF-next-a g*))
| *GNF-next-a* (*next f*) = [*f*]
| *GNF-next-a* (*f schop g*) = ((*map* (λ*x*. *x schop g*) (*GNF-next-a f*)) @ (*GNF-next-a g*))
| *GNF-next-a* (*schopstar f*) = (*map* (λ*x*. *x schop* (*schopstar f*)) (*GNF-next-a f*))
| *GNF-next-a* (*omega f*) = (*map* (λ*x*. *x schop* (*omega f*)) (*GNF-next-a f*))
| *GNF-next-a* (*Ex n. f*) = (*map* (λ*x*. (*Ex n. f*)) (*GNF-next-a f*))

definition *full-system*
where *full-system F* = ((⊢ *big-ior F*) ∧ *mutex F*)

definition *GNF-cons*
where *GNF-cons ae a A* =
((*ae iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *a A*)))

definition *GNF*
where *GNF f* =
(*GNF-cons* (*GNF-empty f*) (*GNF-state f*) (*GNF-next f*))

definition *GNF-prop*
where *GNF-prop X* =
(∃ *ae a A*.
(*state-qpitl ae*) ∧ (*state-qpitl-list a*) ∧
full-system a ∧ *length a* = *length A* ∧
(⊢ *X ieqv GNF-cons ae a A*)
)

definition *GNF-det*
where *GNF-det f* =
(*GNF-cons* (*GNF-empty f*)
(*foldr* (*add-to-now-single*) (*GNF-state f*) [*itrue*])
(*foldr* (*add-to-next-single*) (*GNF-next f*) [*ifalse*])
)

5.3.2 Mutex lemmas

lemma *mutex-single-empty*:
shows *mutex-single f* []
unfolding *mutex-single-def* **by** *auto*

lemma *mutex-single-Cons*:
shows *mutex-single f* (*f1* # *F*) ⟷
(⊢ *inot* (*f iand f1*)) ∧ *mutex-single f F*
unfolding *mutex-single-def* **using** *less-Suc-eq-0-disj* **by** *auto*

lemma *mutex-single-append*:
shows *mutex-single f* (*F1*@ *F2*) ⟷ *mutex-single f F1* ∧ *mutex-single f F2*
proof (*induct F1 arbitrary*: *F2*)
case *Nil*
then show *?case* **unfolding** *mutex-single-def* **by** *simp*
next
case (*Cons a F1*)
then show *?case*
proof −
have *1*: *mutex-single f* ((*a* # *F1*) @ *F2*) ⟷
(⊢ *inot* (*f iand a*)) ∧ *mutex-single f* (*F1* @*F2*)
by (*simp add*: *mutex-single-Cons*)
have *2*: *mutex-single f* (*F1* @*F2*) ⟷ *mutex-single f F1* ∧ *mutex-single f F2*
using *Cons* **by** *blast*
have *3*: (⊢ *inot* (*f iand a*)) ∧ (*mutex-single f F1* ∧ *mutex-single f F2*) ⟷
(*mutex-single f* (*a* # *F1*) ∧ *mutex-single f F2*)
using *mutex-single-Cons* **by** *blast*
show *?thesis*
using *1 2 3* **by** *presburger*
qed
qed

lemma *mutex-empty*:
shows *mutex* []
unfolding *mutex-def* **by** *auto*

lemma *mutex-Cons*:
shows *mutex* (*f* # *F1*) ⟷ *mutex-single f F1* ∧ *mutex F1*
unfolding *mutex-def mutex-single-def*

using *less-Suc-eq-0-disj*
apply *auto*
apply (*metis diff-Suc-1′ nth-Cons′*)
by (*metis diff-Suc-1′ nth-Cons-Suc*)

lemma *mutex-append*:
shows *mutex* (*F1* @ *F2*) ⟷
mutex F1 ∧ *foldr* (λ*x y*. (*mutex-single x F2*) ∧ *y*) *F1 True* ∧ *mutex F2*
proof (*induct F1 arbitrary*: *F2*)
case *Nil*
then show *?case*
by (*simp add*: *mutex-empty*)
next
case (*Cons a F1*)
then show *?case*
proof −
have *1*: *mutex* ((*a* # *F1*) @ *F2*) ⟷
mutex-single a F1 ∧ *mutex-single a F2* ∧ *mutex* (*F1*@*F2*)
by (*simp add*: *mutex-Cons mutex-single-append*)
have *2*: *foldr* (λ*x*. (∧) (*mutex-single x F2*)) (*a* # *F1*) *True* ⟷
mutex-single a F2 ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) (*F1*) *True*
by *simp*
have *3*: *mutex* (*F1* @ *F2*) ⟷ (*mutex F1* ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) *F1 True* ∧ *mutex F2*)
using *Cons* **by** *blast*
have *4*: *mutex-single a F1* ∧ *mutex-single a F2* ∧ *mutex* (*F1*@*F2*) ⟷
mutex-single a F1 ∧ *mutex-single a F2* ∧
(*mutex F1* ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) *F1 True* ∧ *mutex F2*)
using *1 3* **by** *auto*
have *5*: *mutex-single a F1* ∧ *mutex-single a F2* ∧
(*mutex F1* ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) *F1 True* ∧ *mutex F2*) ⟷
(*mutex* (*a* # *F1*) ∧ *foldr* (λ*x*. (∧) (*mutex-single x F2*)) (*a* # *F1*) *True* ∧ *mutex F2*)
using *2 mutex-Cons* **by** *blast*
show *?thesis*
using *1 4 5* **by** *presburger*
qed
qed

lemma *unfold-mutex-single*:
foldr (λ*x y*. (*mutex-single x F2*) ∧ *y*) *F1 True* ⟷
(∀ *i*. *0* ≤ *i* ∧ *i* < *length F1* ⟶ (*mutex-single* (*F1* ! *i*) *F2*))
proof (*induct F1 arbitrary*: *F2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a F1*)
then show *?case*
using *less-Suc-eq-0-disj* **by** *auto*
qed

5.3.3 Add to lemmas

lemma *length-add-to-now-single*:
shows *length* (*add-to-now-single f L*) = (*if L=[] then 0 else 2∗*(*length L*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *add-to-now-single-def* **by** *simp*
next
case (*Cons a L*)
then show *?case* **unfolding** *add-to-now-single-def* **by** *simp*
qed

lemma *length-add-to-next-single*:
shows *length* (*add-to-next-single f L*) = (*if L=[] then 0 else 2∗*(*length L*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *add-to-next-single-def* **by** *simp*
next
case (*Cons a L*)
then show *?case* **unfolding** *add-to-next-single-def* **by** *simp*
qed

lemma *length-add-to-next-single-a*:
shows *length* (*add-to-next-single-a f L*) = (*if L=[] then 0 else 2∗*(*length L*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *add-to-next-single-a-def* **by** *simp*
next
case (*Cons a L*)
then show *?case* **unfolding** *add-to-next-single-a-def* **by** *simp*
qed

lemma *length-add-to-now-next-gen*:
assumes *length xs* = *length ys*
length L1 = *length L2*
shows *length* (*foldr add-to-now-single xs L1*) =
length (*foldr add-to-next-single ys L2*)
using *assms*
proof (*induct xs ys rule*: *list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
using *Cons* **unfolding** *add-to-now-single-def add-to-next-single-def*
by *simp*
qed

lemma *length-add-to-now-next-a-gen*:

assumes *length xs = length ys*
length L1 = length L2
shows *length* (*foldr add-to-now-single xs L1*) =
length (*foldr add-to-next-single-a ys L2*)
using *assms*
proof (*induct xs ys rule*: *list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
using *Cons* **unfolding** *add-to-now-single-def add-to-next-single-a-def*
by *simp*
qed

lemma *length-add-to-now-next*:
assumes *length xs = length ys*
shows *length* (*foldr add-to-now-single xs* [*itrue* ,*ifalse*]) =
length (*foldr add-to-next-single ys* [*ifalse*,*ifalse*])
using *assms*
by (*simp add*: *length-add-to-now-next-gen*)

lemma *length-add-to-now-next-a*:
assumes *length xs = length ys*
shows *length* (*foldr add-to-now-single xs* [*itrue* ,*ifalse*]) =
length (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])
using *assms*
by (*simp add*: *length-add-to-now-next-a-gen*)

lemma *add-to-now-single-to-big-ior-inv*:
assumes ⊢ *big-ior L*
shows ⊢ *big-ior* (*add-to-now-single f L*)
unfolding *add-to-now-single-def* **using** *assms*
big-ior-append[*of* (*map* (λ*x*. (*f iand x*)) *L*) (*map* (λ*x*. (*inot f iand x*)) *L*)]
by (*auto simp add*: *big-ior-defs*)
(*metis iand-defs nth-map*)

lemma *big-ior-foldr-add-to-now-single-inv-a*:
shows ⊢ *big-ior* (*foldr add-to-now-single xs* [*itrue*])
proof (*induct xs*)
case *Nil*
then show *?case*
using *big-ior-defs* **by** *fastforce*
next
case (*Cons a xs*)
then show *?case*
using *Cons add-to-now-single-to-big-ior-inv* **by** *auto*
qed

lemma *big-ior-foldr-add-to-now-single-inv*:
shows $\vdash$ *big-ior* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
proof (*induct xs*)
case *Nil*
then show *?case*
using *big-ior-defs* **by** *fastforce*
next
case (*Cons a xs*)
then show *?case*
using *Cons add-to-now-single-to-big-ior-inv* **by** *auto*
qed

lemma *add-to-now-single-to-mutex-inv*:
assumes *mutex L*
shows *mutex* (*add-to-now-single f L*)
proof −
have *1*: *mutex* (*map* (λx. *f iand x*) *L*)
using *assms* **unfolding** *mutex-def* **by** *auto*
have *2*: *mutex* (*map* (λx. *inot f iand x*) *L*)
using *assms* **unfolding** *mutex-def* **by** *auto*
have *3*: *foldr* (λx. ($\wedge$)) (*mutex-single x* (*map* (λx. (*inot f*) *iand x*) *L*))) (*map* (λx. *f iand x*) *L*) *True*
$\longleftrightarrow$ ($\forall$ *i*. *0* $\leq$ *i* $\wedge$ *i* < *length* (*map* (λx. *f iand x*) *L*) $\longrightarrow$ *mutex-single* (*map* (λx. *f iand x*) *L* ! *i*) (*map* (λx. (*inot f*) *iand x*) *L*))
using *unfold-mutex-single*[*of* (*map* (λx. (*inot f*) *iand x*) *L*) (*map* (λx. *f iand x*) *L*)]
by *blast*
have *4*: ($\forall$ *i*. *0* $\leq$ *i* $\wedge$ *i* < *length* (*map* (λx. *f iand x*) *L*) $\longrightarrow$ *mutex-single* (*map* (λx. *f iand x*) *L* ! *i*) (*map* (λx. (*inot f*) *iand x*) *L*))
using *assms* **unfolding** *mutex-single-def* **by** *auto*
show *?thesis* **unfolding** *add-to-now-single-def* **using** *assms*
mutex-append[*of* (*map* (λx. *f iand x*) *L*) (*map* (λx. (*inot f*) *iand x*) *L*)]
using *1 2 3 4* **by** *blast*
qed

lemma *mutex-foldr-add-to-now-single-inv-a*:
shows *mutex* (*foldr add-to-now-single xs* [*itrue*])
proof (*induct xs*)
case *Nil*
then show *?case* **unfolding** *mutex-def* **by** *simp*
next
case (*Cons a xs*)
then show *?case* **by** (*simp add*: *Cons.hyps add-to-now-single-to-mutex-inv*)
qed

lemma *mutex-foldr-add-to-now-single-inv*:
shows *mutex* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
proof (*induct xs*)
case *Nil*
then show *?case* **unfolding** *mutex-def* **by** *simp*

next
case (*Cons a xs*)
then show *?case* **by** (*simp add*: *Cons.hyps add-to-now-single-to-mutex-inv*)
qed

lemma *add-to-now-single-to-state-qpitl-list*:
assumes *state-qpitl f*
state-qpitl-list L
shows *state-qpitl-list* (*add-to-now-single f L*)
using *assms* **unfolding** *add-to-now-single-def*
by (*simp add*: *inot-d-def state-qpitl-list-append state-qpitl-list-map*)

lemma *state-qpitl-foldr-add-to-now-single-inv-a*:
assumes *state-qpitl-list xs*
shows *state-qpitl-list* (*foldr add-to-now-single xs* [*itrue*])
using *assms*
proof (*induct xs*)
case *Nil*
then show *?case* **using** *Nil*
by (*simp add*: *inot-d-def itrue-d-def state-qpitl-list-def*)
next
case (*Cons a xs*)
then show *?case*
by *simp*
(*metis add-to-now-single-to-state-qpitl-list append-Cons append-Nil length-Cons nth-Cons-0 state-qpitl-list-append state-qpitl-list-defs zero-less-Suc*)
qed

lemma *state-qpitl-foldr-add-to-now-single-inv*:
assumes *state-qpitl-list xs*
shows *state-qpitl-list* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *assms*
proof (*induct xs*)
case *Nil*
then show *?case* **using** *Nil*
by (*simp add*: *inot-d-def itrue-d-def state-qpitl-list-def*)
next
case (*Cons a xs*)
then show *?case*
by *simp*
(*metis add-to-now-single-to-state-qpitl-list append-Cons append-Nil length-Cons nth-Cons-0 state-qpitl-list-append state-qpitl-list-defs zero-less-Suc*)
qed

lemma *det-big-ior-ieqv-nondet-big-ior-a*:
assumes *length L1* = *length L2*
shows ⊢ *big-ior* (*map2* (λ *x y*. *x iand* (*next y*))
(*foldr* (*add-to-now-single*) *L1* [*itrue*])
(*foldr* (*add-to-next-single*) *L2* [*ifalse*]))

```
        ieqv
        big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
   proof (cases xs = [] ∧ ys = [] )
   case True
   then show ?thesis using Cons
    unfolding add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def
     by simp
   next
   case False
   then show ?thesis
     proof −
      have 1: (foldr add-to-now-single (x # xs) [itrue]) =
              ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
         by (simp add: add-to-now-single-def)
      have 2: (foldr add-to-next-single (y # ys) [ifalse]) =
              ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse]))@
                                (foldr add-to-next-single ys [ifalse]) )
        by (simp add: add-to-next-single-def)
      have 3: length xs = length ys
         by (simp add: Cons.hyps(1))
      have 4: ⊢ big-ior (map2 (λx y. x iand next  y)
                        (foldr add-to-now-single xs [itrue])
                        (foldr add-to-next-single ys [ifalse])) ieqv
                big-ior (map2 (λx y. x iand next y) xs ys)
        using 3 Cons.hyps(2) False by fastforce
      have 5: big-ior (map2 (λx y. x iand next  y)
                      (foldr add-to-now-single (x # xs) [itrue])
                      (foldr add-to-next-single (y # ys) [ifalse])) =
              big-ior (map2 (λx y. x iand next  y)
                      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
                      ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse]))@
                                        (foldr add-to-next-single ys [ifalse]) )
                       )
        using 1 2 by presburger
      have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) =
              length (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse]))
        by (simp add: 3 length-add-to-now-next-gen)
      have 7:  length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) =
               length (foldr add-to-next-single ys [ifalse])
         using 6 by auto
      have 75: length (foldr add-to-now-single xs [itrue]) =
```

length (*foldr add-to-next-single ys* [*ifalse*])
using *7* **by** *force*
have *8*: ⊢ *big-ior* (*map2* (λ*x y*. *x iand next y*)
((*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*])) @
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*])))
((*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*]))@
(*foldr add-to-next-single ys* [*ifalse*]))
) *ieqv*
big-ior ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*]))
)
using *6* **by** *force*
have *9*: ⊢ *big-ior* ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*]))
) *ieqv*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*])))
using *big-ior-append* **by** *blast*
have *10*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single ys* [*ifalse*])))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* (*y ior y1*))
((*foldr add-to-now-single xs* [*itrue*]))
((*foldr add-to-next-single ys* [*ifalse*]))))
using *map2-map-a*[*of* (*foldr add-to-now-single xs* [*itrue*])
(*foldr add-to-next-single ys* [*ifalse*]) *x y*] **using** *7* **by** *auto*
have *11*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*]))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. ((*inot x*) *iand x1*) *iand next y1*)
((*foldr add-to-now-single xs* [*itrue*]))
(*foldr add-to-next-single ys* [*ifalse*])))
using *map2-map-b*[*of* (*foldr add-to-now-single xs* [*itrue*])
(*foldr add-to-next-single ys* [*ifalse*]) *x*] *75*
by *auto*
have *12*: ⊢ *big-ior* (*foldr add-to-now-single xs* [*itrue*])

```
      using big-ior-foldr-add-to-now-single-inv-a by metis
    have 13: 0 < length (foldr add-to-now-single xs [itrue])
      using 12 big-ior-empty by auto
    have 14: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single ys [ifalse])))) ior
              (big-ior (map2 (λx y. x iand next y)
                    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                                  (foldr add-to-next-single ys [ifalse]) ))
              ieqv
              (x iand next y) ior
              (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single ys [ifalse]))))
       using big-ior-split-off[of ( (foldr add-to-now-single xs [itrue]))
         ( (foldr add-to-next-single ys [ifalse])) x y] 12 13
      by (simp add: 75 map2-map-b)
    have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))  ieqv
              (x iand next y) ior
               big-ior (map2 (λx y. x iand next y) xs  ys)
       using big-ior-Cons-alt by auto
   have 16: ⊢ (x iand next y) ior
              (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                    ( (foldr add-to-now-single xs [itrue]))
                    ( (foldr add-to-next-single ys [ifalse])))) ieqv
            big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
      using 4 15 by auto
    show ?thesis
    using 10 14 16 5 6 9 by force
  qed
  qed
qed

lemma det-big-ior-ieqv-nondet-big-ior-a-a:
assumes length L1 = length L2
shows ⊢ big-ior ( map2 (λ x y. x iand (next  y))
              (foldr (add-to-now-single) L1 [itrue])
              (foldr (add-to-next-single-a) L2 [ifalse]))
      ieqv
      big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis using Cons
```

```
 unfolding add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def
  by simp
next
case False
then show ?thesis
  proof −
   have 1: (foldr add-to-now-single (x # xs) [itrue]) =
          ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
            (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
      by (simp add: add-to-now-single-def)
   have 2: (foldr add-to-next-single-a (y # ys) [ifalse]) =
          ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
                        (foldr add-to-next-single-a ys [ifalse]) )
     by (simp add: add-to-next-single-a-def)
   have 3: length xs = length ys
      by (simp add: Cons.hyps(1))
   have 4: ⊢ big-ior (map2 (λx y. x iand next  y)
                    (foldr add-to-now-single xs [itrue])
                    (foldr add-to-next-single-a ys [ifalse])) ieqv
             big-ior (map2 (λx y. x iand next y) xs ys)
     using 3 Cons.hyps(2) False by fastforce
   have 5: big-ior (map2 (λx y. x iand next  y)
                  (foldr add-to-now-single (x # xs) [itrue])
                  (foldr add-to-next-single-a (y # ys) [ifalse])) =
           big-ior (map2 (λx y. x iand next  y)
                  ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
                ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
                              (foldr add-to-next-single-a ys [ifalse]) )
                 )
     using 1 2 by presburger
   have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) =
          length (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
     by (simp add: 3 length-add-to-now-next-a-gen)
   have 7:  length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) =
            length (foldr add-to-next-single-a ys [ifalse])
      using 6 by auto
   have 75: length (foldr add-to-now-single xs [itrue]) =
            length (foldr add-to-next-single-a ys [ifalse])
    using 7 by force
   have 8: ⊢ big-ior (map2 (λx y. x iand next y)
                  ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
                    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
                ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
                              (foldr add-to-next-single-a ys [ifalse]) )
                 ) ieqv
              big-ior ( (map2 (λx y. x iand next y)
                        (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                       (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
```

```
                    @
                      (map2 (λx y. x iand next y)
                      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                      (foldr add-to-next-single-a ys [ifalse]) )
                    )
          using 6 by force
      have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
                      (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
                    @
                      (map2 (λx y. x iand next y)
                      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                      (foldr add-to-next-single-a ys [ifalse]) )
                    ) ieqv
                (big-ior  (map2 (λx y. x iand next y)
                      (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)) ior
                (big-ior (map2 (λx y. x iand next y)
                      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                                (foldr add-to-next-single-a ys [ifalse]) ))
        using big-ior-append by blast
      have 10: ⊢ (big-ior  (map2 (λx y. x iand next y)
                      (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)) ieqv
                (big-ior  (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
                      ( (foldr add-to-now-single xs [itrue]))
                      ( (foldr add-to-next-single-a ys [ifalse]))))
          using map2-map-a-a[of (foldr add-to-now-single xs [itrue])
          (foldr add-to-next-single-a ys [ifalse]) x y ] using 7 by auto
      have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
                      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                                (foldr add-to-next-single-a ys [ifalse]) )) ieqv
                (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
                      ( (foldr add-to-now-single xs [itrue]))
                        (foldr add-to-next-single-a ys [ifalse]) ))
        using map2-map-b[of (foldr add-to-now-single xs [itrue])
          (foldr add-to-next-single-a ys [ifalse]) x] 75
        by auto
      have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue])
        using big-ior-foldr-add-to-now-single-inv-a by metis
      have 13: 0 < length (foldr add-to-now-single xs [itrue])
        using 12 big-ior-empty by auto
      have 14: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
                      ( (foldr add-to-now-single xs [itrue]))
                      ( (foldr add-to-next-single-a ys [ifalse])))) ior
                (big-ior (map2 (λx y. x iand next y)
                      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
```

```
                                        (foldr add-to-next-single-a ys [ifalse]) ))
                ieqv
                (x iand next y) ior
                (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                        ( (foldr add-to-now-single xs [itrue]))
                        ( (foldr add-to-next-single-a ys [ifalse]))))
          using big-ior-split-off-a[of ( (foldr add-to-now-single xs [itrue]))
            ( (foldr add-to-next-single-a ys [ifalse])) x y] 12 13
         by (simp add: 75 map2-map-b)
      have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))  ieqv
                (x iand next y) ior
                 big-ior (map2 (λx y. x iand next y) xs  ys)
          using big-ior-Cons-alt by auto
     have 16: ⊢ (x iand next y) ior
                (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                        ( (foldr add-to-now-single xs [itrue]))
                        ( (foldr add-to-next-single-a ys [ifalse])))) ieqv
              big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
        using 4 15 by auto
      show ?thesis
      using 10 14 16 5 6 9 by force
   qed
   qed
qed

lemma det-big-ior-ieqv-nondet-big-ior:
assumes length L1 = length L2
shows ⊢ big-ior ( map2 (λ x y. x iand (next  y))
                (foldr (add-to-now-single) L1 [itrue,ifalse])
                (foldr (add-to-next-single) L2 [ifalse,ifalse]))
        ieqv
        big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
   proof (cases xs = [] ∧ ys = [] )
   case True
   then show ?thesis using Cons
    unfolding add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def
     by simp
   next
   case False
   then show ?thesis
     proof −
      have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
             ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
```

```
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
   by (simp add: add-to-now-single-def)
have 2: (foldr add-to-next-single (y # ys) [ifalse,ifalse]) =
       ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
                    (foldr add-to-next-single ys [ifalse,ifalse]) )
  by (simp add: add-to-next-single-def)
have 3: length xs = length ys
   by (simp add: Cons.hyps(1))
have 4: ⊢ big-ior (map2 (λx y. x iand next  y)
                (foldr add-to-now-single xs [itrue,ifalse])
                (foldr add-to-next-single ys [ifalse,ifalse])) ieqv
         big-ior (map2 (λx y. x iand next y) xs ys)
  using 3 Cons.hyps(2) False by fastforce
have 5: big-ior (map2 (λx y. x iand next  y)
               (foldr add-to-now-single (x # xs) [itrue,ifalse])
               (foldr add-to-next-single (y # ys) [ifalse,ifalse])) =
       big-ior (map2 (λx y. x iand next  y)
               ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
                 (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
               ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
                            (foldr add-to-next-single ys [ifalse,ifalse]) )
                )
  using 1 2 by presburger
have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
       length (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))
  by (simp add: 3 length-add-to-now-next)
have 7:  length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
        length (foldr add-to-next-single ys [ifalse,ifalse])
   using 6 by auto
have 75: length (foldr add-to-now-single xs [itrue, ifalse]) =
        length (foldr add-to-next-single ys [ifalse, ifalse])
 using 7 by force
have 8: ⊢ big-ior (map2 (λx y. x iand next y)
               ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
                 (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
               ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
                            (foldr add-to-next-single ys [ifalse,ifalse]) )
                ) ieqv
          big-ior ( (map2 (λx y. x iand next y)
                    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
                  @
                    (map2 (λx y. x iand next y)
                    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                    (foldr add-to-next-single ys [ifalse,ifalse]) )
                 )
    using 6 by force
have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
                    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
```

```
                @
                  (map2 (λx y. x iand next y)
                  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                  (foldr add-to-next-single ys [ifalse,ifalse]) )
                ) ieqv
            (big-ior  (map2 (λx y. x iand next y)
                  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )) ior
            (big-ior (map2 (λx y. x iand next y)
                  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                                    (foldr add-to-next-single ys [ifalse,ifalse]) ))
   using big-ior-append by blast
have 10: ⊢ (big-ior  (map2 (λx y. x iand next y)
                  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) ) ieqv
           (big-ior  (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
                  ( (foldr add-to-now-single xs [itrue,ifalse]))
                  ( (foldr add-to-next-single ys [ifalse,ifalse]))))
     using map2-map-a[of (foldr add-to-now-single xs [itrue,ifalse])
    (foldr add-to-next-single ys [ifalse,ifalse]) x y ] using 7 by auto
have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
                  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                                    (foldr add-to-next-single ys [ifalse,ifalse]) )) ieqv
           (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
                  ( (foldr add-to-now-single xs [itrue,ifalse]))
                    (foldr add-to-next-single ys [ifalse,ifalse]) ))
   using map2-map-b[of (foldr add-to-now-single xs [itrue,ifalse])
     (foldr add-to-next-single ys [ifalse,ifalse]) x] 75
  by auto
have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue, ifalse])
  using big-ior-foldr-add-to-now-single-inv by metis
have 13: 0 < length (foldr add-to-now-single xs [itrue, ifalse])
  using 12 big-ior-empty by auto
have 14: ⊢ (big-ior  (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
                  ( (foldr add-to-now-single xs [itrue,ifalse]))
                  ( (foldr add-to-next-single ys [ifalse,ifalse]))) ior
            (big-ior (map2 (λx y. x iand next y)
                  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
                                    (foldr add-to-next-single ys [ifalse,ifalse]) ))
           ieqv
           (x iand next y) ior
           (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                  ( (foldr add-to-now-single xs [itrue,ifalse]))
                  ( (foldr add-to-next-single ys [ifalse,ifalse]))))
    using big-ior-split-off[of ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single ys [ifalse,ifalse])) x y] 12 13
   by (simp add: 75 map2-map-b)
have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))  ieqv
           (x iand next y) ior
            big-ior (map2 (λx y. x iand next y) xs  ys)
```

```
        using big-ior-Cons-alt by auto
    have 16: ⊢ (x iand next y) ior
                (big-ior  (map2 (λx1 y1. x1 iand next  y1)
                          ( (foldr add-to-now-single xs [itrue,ifalse]))
                          ( (foldr add-to-next-single ys [ifalse,ifalse])))) ieqv
              big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
       using 4 15 by auto
     show ?thesis
     using 10 14 16 5 6 9 by force
  qed
  qed
qed

lemma det-big-ior-ieqv-nondet-big-ior-b:
assumes length L1 = length L2
shows ⊢ big-ior ( map2 (λ x y. x iand (next  y))
                (foldr (add-to-now-single) L1 [itrue,ifalse])
                (foldr (add-to-next-single-a) L2 [ifalse,ifalse]))
       ieqv
       big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis using Cons
   unfolding add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def
    by simp
  next
  case False
  then show ?thesis
    proof −
     have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
             ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
               (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
        by (simp add: add-to-now-single-def)
     have 2: (foldr add-to-next-single-a (y # ys) [ifalse,ifalse]) =
            ( (map (λx1. (if y =ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse,ifalse]))@
                          (foldr add-to-next-single-a ys [ifalse,ifalse]) )
       by (simp add: add-to-next-single-a-def)
     have 3: length xs = length ys
        by (simp add: Cons.hyps(1))
     have 4: ⊢ big-ior (map2 (λx y. x iand next  y)
                        (foldr add-to-now-single xs [itrue,ifalse])
                        (foldr add-to-next-single-a ys [ifalse,ifalse])) ieqv
               big-ior (map2 (λx y. x iand next y) xs ys)
```

using *3 Cons.hyps(2) False* **by** *fastforce*
have *5*: *big-ior* (*map2* (λ*x y*. *x iand next y*)
(*foldr add-to-now-single* (*x # xs*) [*itrue*,*ifalse*])
(*foldr add-to-next-single-a* (*y # ys*) [*ifalse*,*ifalse*])) =
big-ior (*map2* (λ*x y*. *x iand next y*)
((*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) @
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])))
((*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))@
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
)
using *1 2* **by** *presburger*
have *6*: *length* (*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) =
length (*map* (λ*x1*. *y ior x1*) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
by (*simp add*: *3 length-add-to-now-next-a*)
have *7*: *length* (*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) =
length (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])
using *6* **by** *auto*
have *75*: *length* (*foldr add-to-now-single xs* [*itrue*, *ifalse*]) =
length (*foldr add-to-next-single-a ys* [*ifalse*, *ifalse*])
using *7* **by** *force*
have *8*: ⊢ *big-ior* (*map2* (λ*x y*. *x iand next y*)
((*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])) @
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*])))
((*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))@
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
) *ieqv*
big-ior ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
)
using *6* **by** *force*
have *9*: ⊢ *big-ior* ((*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
@
(*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))
) *ieqv*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y = ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys*

[*ifalse*,*ifalse*])))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
using *big-ior-append* **by** *blast*
have *10*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. *x iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*map* (λ*x1*. (*if y* = *ifalse then x1 else y ior x1*)) (*foldr add-to-next-single-a ys*
[*ifalse*,*ifalse*])))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* ((*if y* = *ifalse then y1 else y ior y1*)))
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))))
using *map2-map-a-a*[*of* (*foldr add-to-now-single xs* [*itrue*,*ifalse*])
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]) *x y*] **using** *7* **by** *auto*
have *11*: ⊢ (*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))) *ieqv*
(*big-ior* (*map2* (λ*x1 y1*. ((*inot x*) *iand x1*) *iand next y1*)
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
using *map2-map-b*[*of* (*foldr add-to-now-single xs* [*itrue*,*ifalse*])
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]) *x*] *75*
by *auto*
have *12*: ⊢ *big-ior* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *big-ior-foldr-add-to-now-single-inv* **by** *metis*
have *13*: *0* < *length* (*foldr add-to-now-single xs* [*itrue*, *ifalse*])
using *12 big-ior-empty* **by** *auto*
have *14*: ⊢ (*big-ior* (*map2* (λ*x1 y1*. (*x iand x1*) *iand next* (*if y*= *ifalse then y1 else y ior y1*))
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))) *ior*
(*big-ior* (*map2* (λ*x y*. *x iand next y*)
(*map* (λ*x1*. (*inot x*) *iand x1*) (*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
(*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))
ieqv
(*x iand next y*) *ior*
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next y1*)
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*]))))
using *big-ior-split-off-a*[*of* ((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])) *x y*] *12 13*
by (*simp add*: *75 map2-map-b*)
have *15*: ⊢ *big-ior* (*map2* (λ*x y*. *x iand next y*) (*x* # *xs*) (*y* # *ys*)) *ieqv*
(*x iand next y*) *ior*
big-ior (*map2* (λ*x y*. *x iand next y*) *xs ys*)
using *big-ior-Cons-alt* **by** *auto*
have *16*: ⊢ (*x iand next y*) *ior*
(*big-ior* (*map2* (λ*x1 y1*. *x1 iand next y1*)
((*foldr add-to-now-single xs* [*itrue*,*ifalse*]))
((*foldr add-to-next-single-a ys* [*ifalse*,*ifalse*])))) *ieqv*
big-ior (*map2* (λ*x y*. *x iand next y*) (*x* # *xs*) (*y* # *ys*))

using *4 15* **by** *auto*
show *?thesis*
using *10 14 16 5 6 9* **by** *force*
qed
qed
qed

lemma *mutex-single-filter-sat*:
assumes *mutex-single f L1*
shows *mutex-single f* (*filter* (λ*x*. (∃ *σ*. *σ* ⊨ *x*)) *L1*)
using *assms*
proof (*induct L1*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L1*)
then show *?case*
by (*simp add*: *mutex-single-Cons*)
qed

lemma *mutex-filter-sat*:
assumes *mutex L1*
shows *mutex* (*filter* (λ*x*. (∃ *σ*. *σ* ⊨ *x*)) *L1*)
using *assms*
proof (*induct L1*)
case *Nil*
then show *?case* **unfolding** *mutex-def* **by** *simp*
next
case (*Cons a L1*)
then show *?case*
by (*simp add*: *mutex-Cons mutex-single-filter-sat*)
qed

lemma *state-qpitl-list-filter-sat*:
assumes *state-qpitl-list L*
shows *state-qpitl-list* (*filter* (λ*x*. (∃ *σ*. *σ* ⊨ *x*)) *L*)
using *assms*
proof (*induct L*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L*)
then show *?case*
by (*simp add*: *state-qpitl-list-def*)
qed

lemma *big-ior-itrue*:
$\vdash$ *big-ior* [*itrue*]
by (*simp add*: *big-ior-defs*)

lemma *mutex-itrue*:
mutex [*itrue*]
by (*simp add*: *mutex-Cons mutex-empty mutex-single-empty*)

lemma *big-ior-mutex-start*:
($\vdash$ *big-ior* [*ifalse*,*itrue*]) $\wedge$ *mutex* [*ifalse*,*itrue*]
unfolding *big-ior-def mutex-def*
using *less-Suc-eq* **by** *force*

5.3.4 GNF lemmas

lemma *GNF-defs*:
assumes $\vdash$ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-qpitl F-e
state-qpitl-list F-now
length F-now = *length F-next*
shows ($\sigma \models F$) $\longleftrightarrow$
((*nlength* σ = *0* $\wedge$ ((*NNil* (*nfirst* σ)) $\models$ *F-e*)) $\vee$
(*nlength* σ > *0* $\wedge$
($\exists$ *i*<*length F-now*. ((*NNil* (*nfirst* σ)) $\models$ *F-now* !*i*) $\wedge$ ((*ndropn 1* σ) $\models$ *F-next* !*i*))))
proof −
have *1*: ($\sigma \models F$) $\longleftrightarrow$
($\sigma \models$ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))))

using *assms itl-ieq* **by** *blast*
have *2*: ($\sigma \models$ (*F-e iand empty*)) $\longleftrightarrow$
(*nlength* σ = *0* $\wedge$ ((*NNil* (*nfirst* σ)) $\models$ *F-e*))
using *assms state-qpitl-defs empty-defs iand-defs* **by** *blast*
have *3*: *length* (*map2* (λ*x y*. *x iand next y*) *F-now F-next*) = *length F-now*
by (*simp add*: *assms*(*4*))
have *4*:
($\sigma \models$ (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) $\longleftrightarrow$
($\exists$ *i*<*length F-now*. ($\sigma \models$ (λ *x y*. *x iand* (*next y*)) (*F-now*!*i*) (*F-next*!*i*)))
by (*simp add*: *assms*(*4*) *big-ior-map2-defs*)
have *5*: ($\exists$ *i*<*length F-now*. ($\sigma \models$ (λ *x y*. *x iand* (*next y*)) (*F-now*!*i*) (*F-next*!*i*))) $\longleftrightarrow$
(*nlength* σ > *0* $\wedge$
($\exists$ *i*<*length F-now*. ((*NNil* (*nfirst* σ)) $\models$ *F-now* !*i*) $\wedge$ ((*ndropn 1* σ) $\models$ *F-next* !*i*)))
by *simp*
(*metis assms*(*3*) *co.enat.exhaust-sel enat-le-plus-same*(*2*) *not-one-le-zero plus-1-eSuc*(*2*)
state-qpitl-defs state-qpitl-list-defs)
show *?thesis*
using *1 2 4 5* **by** *auto*
qed

lemma *big-ior-iand-next-big-iand-iimp*:
assumes ($\vdash$ *big-ior F*)
mutex F
length F = length G
shows $\vdash$ *big-ior* (*map2* ($\lambda x\ y.\ x$ *iand* (*next y*)) *F G*) *ieqv*
big-iand (*map2* ($\lambda x\ y.\ x$ *iimp* (*next y*)) *F G*)
using *assms* **unfolding** *valid-def mutex-def*
apply (*auto simp add*: *big-ior-defs big-iand-defs map2-iand-next-defs*)
apply (*metis not-less-iff-gr-or-eq*)
by *meson*

lemma *inot-big-iand-next-iimp-big-iand-iimp-inot*:
assumes ($\vdash$ *big-ior F*)
mutex F
length F = length G
shows $\vdash$ *more iimp* (*inot big-iand* (*map2* ($\lambda x\ y.\ x$ *iimp* (*next y*)) *F G*) *ieqv*
big-iand (*map2* ($\lambda\ x\ y.\ x$ *iimp* (*next* (*inot y*))) *F G*))
using *assms* **unfolding** *mutex-def*
apply (*auto simp add*: *big-iand-defs big-ior-defs more-d-def*)
apply (*metis linorder-less-linear*)
by *meson*

lemma *inot-big-ior-iand-big-ior-iand-inot*:
assumes ($\vdash$ *big-ior F*)
mutex F
length F = length G
shows $\vdash$ (*inot* (*big-ior* (*map2* ($\lambda x\ y.\ x$ *iand* (*y*)) *F G*)) *ieqv*
big-ior (*map2* ($\lambda\ x\ y.\ x$ *iand* ((*inot y*))) *F G*))
using *assms* **unfolding** *mutex-def*
apply (*auto simp add*: *big-ior-defs big-iand-defs map2-iand-inot-defs*)
apply *meson*
by (*metis not-less-iff-gr-or-eq*)

lemma *inot-big-ior-iand-big-ior-iand-next-inot*:
assumes ($\vdash$ *big-ior F*)
mutex F
length F = length G
shows $\vdash$ *more iimp* (*inot* (*big-ior* (*map2* ($\lambda x\ y.\ x$ *iand* (*next y*)) *F G*)) *ieqv*
big-ior (*map2* ($\lambda\ x\ y.\ x$ *iand* (*next* (*inot y*))) *F G*))
using *assms* **unfolding** *mutex-def*
apply (*auto simp add*: *big-ior-defs big-iand-defs map2-iand-inot-next-defs more-d-def*)
apply (*metis One-nat-def gr-zeroI map2-iand-inot-next-defs not-one-le-zero semantics-qpitl.simps*(*4*))
by (*metis linorder-less-linear*)

lemma *more-iimp-inot-big-ior-big-ior-map-inot*:
assumes ($\vdash$ *big-ior F*)
mutex F

```
        length F = length G
shows ⊢ more iimp
        ((inot (big-ior (map2 (λx y. x iand (next y)) F G))) ieqv
         big-ior (map2 (λx y. x iand (next y)) F (map (λx. (inot x)) G)))
proof −
 have 1:  (map2 (λx y. x iand (next y)) F (map (λx. (inot x)) G)) =
           (map2 (λx y. x iand (next (inot y))) F G)
       using map2-map[of F G (λx y. x iand (next y)) id (λx. (inot x))]
        assms by auto
 show ?thesis
 using 1 assms inot-big-ior-iand-big-ior-iand-next-inot by auto
qed




lemma GNF-iimp-step:
 assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))
        state-qpitl F-e
        state-qpitl-list F-now
        length F-now = length F-next
        ⊢ G ieqv (( G-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  G-now G-next)))
        state-qpitl G-e
        state-qpitl-list G-now
        length G-now = length G-next
 shows  (σ ⊨ F iimp G) ⟷
        (σ ⊨ ( ( (inot F-e) ior G-e) iand empty) ior
              (big-ior
               (map2 (λ x y. x iand (next y))
                     ((add-to-now F-now ) @G-now)
                     ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) )))
proof (cases nlength σ = 0)
case True
then show ?thesis
  proof −
   have 1: (σ ⊨ F iimp G) ⟷ (¬(σ ⊨ F) ∨ (σ ⊨G))
     by auto
   have 2: (¬(σ ⊨ F) ∨ (σ ⊨G)) ⟷
           (¬((NNil (nfirst σ)) ⊨ F-e) ∨ ((NNil (nfirst σ)) ⊨ G-e))
    using
     assms True
     by (metis (mono-tags, lifting) GNF-defs less-numeral-extra(3))
   have 3: state-qpitl (inot F-e ior G-e)
     by (simp add: assms(2) assms(6) inot-d-def ior-d-def)
   have 35: state-qpitl-list [ifalse, itrue]
      unfolding state-qpitl-list-def itrue-d-def inot-d-def by auto
   have 4: state-qpitl-list ((add-to-now F-now ) @ G-now)
      by (simp add: add-to-now-def assms(3) assms(7) state-qpitl-foldr-add-to-now-single-inv-a
        state-qpitl-list-append)
   have 5: length (add-to-now F-now  @ G-now) =
```

```
        length (map inot-d (add-to-next F-next ) @ G-next)
    by (simp add: add-to-next-def add-to-now-def assms(4) assms(8) length-add-to-now-next-gen)
  have 6: (¬((NNil (nfirst σ)) ⊨ F-e) ∨ ((NNil (nfirst σ)) ⊨ G-e)) ⟷
        (σ ⊨ ( ( (inot F-e) ior G-e) iand empty) ior
          (big-ior
           (map2 (λ x y. x iand (next y))
             ((add-to-now F-now )@G-now)
             ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) ) ))
    using GNF-defs[of
        ( ( (inot F-e) ior G-e) iand empty) ior
        (big-ior
         (map2 (λ x y. x iand (next y))
         ((add-to-now F-now )@G-now)
         ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) ))
          ( (inot F-e) ior G-e)
                ((add-to-now F-now )@G-now)
                ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) σ ]
    using assms
    using 3 4 5 True by force
  show ?thesis
  using 1 2 6 by blast
 qed
next
case False
then show ?thesis
 proof −
  have 8: (σ ⊨ F iimp G) ⟷ (σ ⊨ (inot F) ior G)
    by auto
  have 9: (σ ⊨ (inot F)) ⟷
       (σ ⊨ (inot (big-ior (map2 (λ x y. x iand (next y))  F-now F-next))))
   using False assms(1) by auto
  have 10: ⊢ (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)) ieqv
         big-ior ( map2 (λ x y. x iand (next  y))
             (foldr (add-to-now-single) F-now [itrue])
             (foldr (add-to-next-single) F-next [ifalse]))
   using assms(4) det-big-ior-ieqv-nondet-big-ior-a by auto
  have 101: (σ ⊨ (inot (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))) ⟷
        (σ ⊨ (inot (big-ior ( map2 (λ x y. x iand (next  y))
             (foldr (add-to-now-single) F-now [itrue])
             (foldr (add-to-next-single) F-next [ifalse])))))
    using 10 by auto
  have 11: ⊢ big-ior (foldr (add-to-now-single) F-now [itrue])
   using big-ior-foldr-add-to-now-single-inv-a by blast
  have 12: mutex (foldr (add-to-now-single) F-now [itrue])
   using mutex-foldr-add-to-now-single-inv-a by blast
  have 13: length (foldr (add-to-now-single) F-now [itrue]) =
       length (foldr (add-to-next-single) F-next [ifalse])
   by (simp add: assms(4) length-add-to-now-next-gen)
  have 14: ⊢ more iimp
         ( (inot (big-ior ( map2 (λ x y. x iand (next  y))
```

```
                (foldr (add-to-now-single) F-now [itrue])
                (foldr (add-to-next-single) F-next [ifalse]))))) ieqv
            (big-ior
            (map2 (λ x y. x iand (next y))
                (foldr (add-to-now-single) F-now [itrue])
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))))))
    using 11 12 13 more-iimp-inot-big-ior-big-ior-map-inot by blast
  have 15: (σ ⊨ (inot (big-ior ( map2 (λ x y. x iand (next  y))
                (foldr (add-to-now-single) F-now [itrue])
                (foldr (add-to-next-single) F-next [ifalse])))) ⟷
        (σ ⊨ (big-ior
            (map2 (λ x y. x iand (next y))
                (foldr (add-to-now-single) F-now [itrue])
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))))))
     using 14 False by auto
     (metis (no-types, lifting) empty-d-def empty-defs inot-defs,
      metis (no-types, lifting) empty-d-def empty-defs inot-defs)
  have 16: (σ ⊨ G) ⟷
          (σ ⊨ (big-ior (map2 (λ x y. x iand (next y))  G-now G-next)))
   using False assms(5) by auto
  have 17:  (σ ⊨ (inot F) ior G) ⟷
          (σ ⊨ (big-ior  (map2 (λ x y. x iand (next y))
                (foldr (add-to-now-single) F-now [itrue])
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))) ior
              (big-ior (map2 (λ x y. x iand (next y))  G-now G-next)))
              using 101 15 16 9 by auto
  have 18: (σ ⊨ (big-ior  (map2 (λ x y. x iand (next y))
                (foldr (add-to-now-single) F-now [itrue])
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))) ior
              (big-ior (map2 (λ x y. x iand (next y))  G-now G-next))) ⟷
        (σ ⊨ (big-ior  (map2 (λ x y. x iand (next y))
                ((foldr (add-to-now-single) F-now [itrue]) @ G-now)
                ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])) @ G-next) ) ))
  using 13 big-ior-append by auto
  show ?thesis unfolding add-to-now-def add-to-next-def
  using 17 18 False by auto
 qed
qed




lemma GNF-inot-step:
 assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))
        state-qpitl F-e
        state-qpitl-list F-now
        length F-now = length F-next
shows  (σ ⊨ (inot F) ) ⟷
        (σ ⊨ (  (inot F-e) iand empty) ior
            (big-ior
```

```
              (map2 (λ x y. x iand (next y))
                   ((add-to-now F-now ) @[ifalse])
                   ((map (λx. (inot x)) (add-to-next F-next )) @ [ifalse]) )))
proof −
 have 1: (σ ⊨ (inot F) ) ⟷ (σ ⊨ F iimp ifalse)
   by auto
 have 2: ⊢ ifalse ieqv ifalse iand empty ior big-ior (map2 (λx y. x iand next y) [ifalse] [ifalse])
    by (auto simp add: big-ior-defs)
 have 3: (σ ⊨ F iimp ifalse) ⟷
        (σ ⊨ ( ( (inot F-e) ior ifalse) iand empty) ior
             (big-ior
              (map2 (λ x y. x iand (next y))
                   ((add-to-now F-now ) @ [ifalse])
                   ((map (λx. (inot x)) (add-to-next F-next )) @ [ifalse]) )))
    using GNF-iimp-step[of F F-e F-now F-next ifalse ifalse [ifalse] [ifalse] σ]
     assms
    by (metis 2 length-Cons less-Suc0 list.size(3) nth-Cons-0 state-qpitl.simps(1)
       state-qpitl-list-defs)
 show ?thesis
 using 3 by auto
qed

lemma GNF-inot-step-valid:
 assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))
        state-qpitl F-e
        state-qpitl-list F-now
        length F-now = length F-next
shows ⊢ (inot F)  ieqv
        ( (  (inot F-e) iand empty) ior
             (big-ior
              (map2 (λ x y. x iand (next y))
                   ((add-to-now F-now ) @[ifalse])
                   ((map (λx. (inot x)) (add-to-next F-next )) @ [ifalse]) )))
using assms GNF-inot-step[of F F-e F-now F-next ]
using itl-ieq by blast



lemma GNF-inot-inot-step-valid:
 assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))
        state-qpitl F-e
        state-qpitl-list F-now
        length F-now = length F-next
shows ⊢ (inot (inot F))  ieqv
        ( (  F-e iand empty) ior
             (big-ior  (map2 (λ x y. x iand (next y)) F-now F-next )))
using assms by simp
```

lemma *GNF-ior-step*:
assumes $\vdash$ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-qpitl F-e
state-qpitl-list F-now
length F-now = *length F-next*
$\vdash$ *G ieqv* ((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*)))
state-qpitl G-e
state-qpitl-list G-now
length G-now = *length G-next*
shows ($\sigma \models$ *F ior G*) $\longleftrightarrow$
($\sigma \models$ ((*F-e ior G-e*) *iand empty*) *ior*
(*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) (*F-now* @ *G-now*) (*F-next* @ *G-next*))))
proof −
have *1*: ($\sigma \models$ *F ior G*) $\longleftrightarrow$
($\sigma \models$ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) *ior*
((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*))))
using *assms* **by** *auto*
have *2*: ($\sigma \models$ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) *ior*
((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*)))) $\longleftrightarrow$
($\sigma \models$ ((*F-e ior G-e*) *iand empty*) *ior*
(*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) (*F-now* @ *G-now*) (*F-next* @ *G-next*))))
using *assms big-ior-map2-append* **by** *fastforce*
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *GNF-schop-step*:
assumes $\vdash$ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
state-qpitl F-e
state-qpitl-list F-now
length F-now = *length F-next*
$\vdash$ *G ieqv* ((*G-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *G-now G-next*)))
state-qpitl G-e
state-qpitl-list G-now
length G-now = *length G-next*
shows ($\sigma \models$ *F schop G*) $\longleftrightarrow$
($\sigma \models$ (*F-e iand G-e*) *iand empty ior*
(*big-ior* (*map2* (λ *x y*. *x iand next y*)
(*F-now* @ (*map* (λ*x*. *x iand F-e*) *G-now*))
((*map* (λ*x*. *x schop G*) *F-next*) @ *G-next*))))
proof (*cases nlength* σ = *0*)
case *True*
then show *?thesis*
proof −
have *1*: ($\sigma \models$ *F schop G*) $\longleftrightarrow$ ($\sigma \models$ *F*) $\wedge$ ($\sigma \models$ *G*)
using *True* **by** (*auto simp add*: *ntaken-all zero-enat-def*)
have *2*: ($\sigma \models$ *F*) $\wedge$ ($\sigma \models$ *G*) $\longleftrightarrow$ ((*NNil* (*nfirst* σ)) $\models$ *F-e*) $\wedge$ ((*NNil* (*nfirst* σ)) $\models$ *G-e*)
using *GNF-defs True assms*
by (*metis less-numeral-extra*(*3*))

```
  have 3: state-qpitl (F-e iand G-e)
    by (simp add: assms(2) assms(6) iand-d-def inot-d-def ior-d-def)
  have 4: state-qpitl-list (F-now @ map (λx. x iand F-e) G-now)
    using assms state-qpitl-list-append state-qpitl-list-map-alt by blast
  have 5: length (F-now @ map (λx. x iand F-e) G-now) =
          length (map (λx.  x schop G ) F-next @ G-next)
     by (simp add: assms(4) assms(8))
  have 6: ((NNil (nfirst σ)) ⊨ F-e) ∧ ((NNil (nfirst σ)) ⊨ G-e) ⟷
          (σ ⊨ (F-e iand G-e) iand empty ior
              (big-ior (map2 (λ x y. x iand next y)
                       (F-now @ (map (λx. x iand F-e) G-now))
                       ((map (λx. x schop G) F-next) @  G-next))) )
     using GNF-defs[of (F-e iand G-e) iand empty ior
              (big-ior (map2 (λ x y. x iand next y)
                       (F-now @ (map (λx. x iand F-e) G-now))
                       ((map (λx. x schop G) F-next) @  G-next)))
              (F-e iand G-e) (F-now @ (map (λx. x iand F-e) G-now))
              ((map (λx. x schop G) F-next) @  G-next) σ]
    using assms
    by (simp add: 3 4 True)
  show ?thesis
  using 1 2 6 by presburger
 qed
next
case False
 assume a: nlength σ ≠ 0
 show ?thesis (is ?lhs ⟷ ?rhs)
  proof
   assume l:?lhs
   show ?rhs
   proof −
    have 7: (∃ i. enat i ≤ nlength σ ∧ (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G))
     using l by simp
    obtain i where 8: enat i ≤ nlength σ ∧ (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G)
      using 7 by blast
    have 9: i=0 ⟹ ?rhs
      proof −
       assume b:i=0
       have 10: (ntaken 0 σ ⊨ F) ∧ (σ ⊨ G)
         using 8 b by auto
       have 11:  ( σ ⊨ F-e)
          using assms
          by (metis (mono-tags, lifting) 8 GNF-defs  b less-numeral-extra(3) min-def ntaken-nfirst
              ntaken-nlength state-qpitl-defs zero-enat-def)
       have 12: σ ⊨ F-e iand big-ior (map2 (λ x y. x iand next y) G-now G-next)
          using 10 11 a assms(5) by force
       have 13: σ ⊨ big-ior (map2 (λ x y. x iand next y)
                       (map (λx. x iand F-e) G-now) G-next)
         using iand-big-ior-dist 12 assms(8) itl-ieq by blast
       show ?thesis
```

```
      using 13 assms(4) assms(8) big-ior-map2-append by force
     qed
   have 14: 0<i ∧ enat i ≤ nlength σ ⟹ ?rhs
    proof −
     assume c: 0<i ∧ enat i ≤ nlength σ
    have 15:  (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G)
      by (simp add: 8)
    have 16:  (ntaken i σ ⊨ (big-ior (map2 (λ x y. x iand next y) F-now F-next)))
      using 8 assms(1) c enat-0-iff(1) by auto
    have 17: ((ntaken i σ) ⊨ (big-ior (map2 (λ x y. x iand next y) F-now F-next))) ⟷
          (∃ ia<length F-now. ntaken i σ ⊨ F-now ! ia iand next F-next ! ia)
      using 16 big-ior-map2-defs[of F-now F-next (ntaken i σ) (λ x y. x iand next y)  ]
      assms by blast
    have 18: length F-now = length (map (λx. x schop G) F-next)
      using assms(4) by fastforce
    have 19: (σ ⊨ (big-ior (map2 (λ x y. x iand next y)
                  F-now  (map (λx. x schop G) F-next) ))) ⟷
          (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i)
      using 18  big-ior-map2-defs[of F-now (map (λx. x schop G) F-next) σ (λ x y. x iand next y)]
    by blast
    obtain j where  20: j<length F-now ∧ (ntaken i σ ⊨ F-now ! j iand next F-next ! j)
      using 16 17 by blast
    have 21: enat (i−1) ≤ nlength σ − enat (Suc 0)
      by (metis One-nat-def c enat-minus-mono1 idiff-enat-enat)
    have 22: ( (ntaken (i−1) (ndropn (Suc 0) σ) ⊨ F-next ! j))
     by (metis 20 One-nat-def Suc-leI c diff-add iand-defs ntaken-ndropn-swap plus-enat-simps(1)
        semantics-qpitl.simps(4))
    have 23: (  (ndropn (i−1) (ndropn (Suc 0) σ) ⊨ G))
     using 8 by (simp add:  c ndropn-ndropn)
    have 24: ( σ ⊨ F-now ! j iand next ((F-next ! j) schop G ) )
      using 20 15 by simp
     (metis 21 22 23 assms(3) ntaken-nfirst state-qpitl-defs state-qpitl-list-defs)
    have 25: (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop G ) F-next ! i)
        using 20 24 assms(4) by fastforce
    show ?thesis
    using 18 19 25 assms(8) big-ior-map2-append by auto
   qed
  show ?thesis
  using 14 8 9 by blast
 qed
 next
  assume r: ?rhs
  show ?lhs
   proof −
    have 26: σ ⊨ big-ior (map2 (λx y. x iand next y) (F-now @ map (λx. x iand F-e) G-now) (map (λx.
x schop G ) F-next @ G-next))
      using a r by auto
    have 27: length F-now = length (map (λx.  x schop G ) F-next)
      by (simp add: assms(4))
    have 28: length (map (λx. x iand F-e) G-now) = length (G-next)
```

using *assms(8)* **by** *fastforce*
have *29*: $(\sigma \models (\textit{big-ior}\ (\textit{map2}\ (\lambda x\ y.\ x\ \textit{iand next}\ y)\ (\textit{F-now})\ (\textit{map}\ (\lambda x.\ \ x\ \textit{schop}\ G\ )\ \textit{F-next}))))$
$\vee$
$(\sigma \models\ \ ((\textit{big-ior}\ (\textit{map2}\ (\lambda x\ y.\ x\ \textit{iand next}\ y)\ (\textit{map}\ (\lambda x.\ x\ \textit{iand F-e})\ \textit{G-now})\ (\textit{G-next})\ \ \ )))$
using *26 27 28 big-ior-map2-append itl-ieq ior-defs* **by** *blast*
have *30*: $(\sigma \models (\textit{big-ior}\ (\textit{map2}\ (\lambda x\ y.\ x\ \textit{iand next}\ y)\ (\textit{F-now})\ (\textit{map}\ (\lambda x.\ \ x\ \textit{schop}\ G\ )\ \textit{F-next})))) \longleftrightarrow$
$(\exists\, i{<}\textit{length F-now}.\ \sigma \models \textit{F-now}\ !\ i\ \textit{iand next map}\ (\lambda x.\ \ x\ \textit{schop}\ G\ )\ \textit{F-next}\ !\ i)$
using *big-ior-map2-defs*[*of F-now* (*map* ($\lambda x.\ \ x$ *schop* G) *F-next*) σ ($\lambda x\ y.\ x$ *iand next* y)]
using *27* **by** *fastforce*
have *31*: $(\exists\, i{<}\textit{length F-now}.\ \sigma \models \textit{F-now}\ !\ i\ \textit{iand next map}\ (\lambda x.\ \ x\ \textit{schop}\ G\ )\ \textit{F-next}\ !\ i) \Longrightarrow \textit{?lhs}$
proof −
assume *d*: $(\exists\, i{<}\textit{length F-now}.\ \sigma \models \textit{F-now}\ !\ i\ \textit{iand next map}\ (\lambda x.\ \ x\ \textit{schop}\ G\ )\ \textit{F-next}\ !\ i)$
obtain *i* **where** *32*: $i{<}\textit{length F-now} \wedge (\sigma \models \textit{F-now}\ !\ i\ \textit{iand next map}\ (\lambda x.\ \ x\ \textit{schop}\ G\ )\ \textit{F-next}\ !\ i)$
using *d* **by** *blast*
have *33*: $(\sigma \models \textit{F-now}\ !\ i)$
using *32 iand-defs* **by** *blast*
have *34*: *map* ($\lambda x.\ \ x$ *schop* G) *F-next* ! i = (*F-next* !i) *schop* G
using *32 assms(4)* **by** *auto*
have *35*: $1 \le \textit{nlength}\ \sigma \wedge (\exists\, \textit{ia}.\ \textit{enat ia} \le \textit{nlength}\ \sigma - \textit{enat}\ (\textit{Suc 0}) \wedge (\textit{ntaken ia}\ (\textit{ndropn}\ (\textit{Suc 0})\ \sigma) \models \textit{F-next}\ !\ i) \wedge (\textit{ndropn ia}\ (\textit{ndropn}\ (\textit{Suc 0})\ \sigma) \models G))$
using *32 34* **by** *simp*
obtain *ia* **where** *36*: $\textit{enat ia} \le \textit{nlength}\ \sigma - \textit{enat}\ (\textit{Suc 0}) \wedge (\textit{ntaken ia}\ (\textit{ndropn}\ (\textit{Suc 0})\ \sigma) \models \textit{F-next}\ !\ i) \wedge (\textit{ndropn ia}\ (\textit{ndropn}\ (\textit{Suc 0})\ \sigma) \models G)$
using *35* **by** *blast*
have *37*: $(\textit{ntaken}\ (\textit{ia+1})\ (\ \sigma) \models (\textit{big-ior}\ (\textit{map2}\ (\lambda\ x\ y.\ x\ \textit{iand}\ (\textit{next}\ y))\ \ \textit{F-now F-next}))) \longleftrightarrow$
$(\exists\, i{<}\textit{length F-now}.\ \textit{ntaken}\ (\textit{ia+1})\ (\ \sigma) \models \textit{F-now}\ !\ i\ \textit{iand next F-next}\ !\ i)$
using *assms big-ior-map2-defs*[*of F-now F-next* (*ntaken* (*ia+1*) (σ)) (λ x y. x *iand* (*next* y))] **by** *blast*
have *38*: $(\textit{ntaken}\ (\textit{ia+1})\ \ \sigma \models \textit{F-now}\ !\ i\ \textit{iand next F-next}\ !\ i)$
by *simp*
(*metis 32 33 35 36 One-nat-def add.commute assms(3) enat-min-eq enat-ord-simps(1) le-add1 ntaken-ndropn-swap ntaken-nfirst one-enat-def plus-1-eq-Suc state-qpitl-defs state-qpitl-list-defs*)
have *39*: $(\textit{ntaken}\ (\textit{ia+1})\ \ \sigma \models F)$
using *32 37 38 assms(1)* **by** *auto*
have *40*: $(\textit{ndropn}\ (\textit{ia+1})\ \ \sigma) \models G$
by (*metis 36 One-nat-def add.commute ndropn-ndropn*)
have *41*: $(\textit{ia+1}) \le \textit{nlength}\ \sigma$
by (*metis 35 36 One-nat-def enat-min-eq one-enat-def plus-enat-simps(1)*)
show *?thesis*
using *39 40 41* **by** *auto*
qed
have *42*: $(\sigma \models\ \ ((\textit{big-ior}\ (\textit{map2}\ (\lambda x\ y.\ x\ \textit{iand next}\ y)\ (\textit{map}\ (\lambda x.\ x\ \textit{iand F-e})\ \textit{G-now})\ (\textit{G-next})\ \ \ ))) \longleftrightarrow$
$(\exists\, i{<}\textit{length}\ (\textit{map}\ (\lambda x.\ x\ \textit{iand F-e})\ \textit{G-now}).\ \sigma \models \textit{map}\ (\lambda x.\ x\ \textit{iand F-e})\ \textit{G-now}\ !\ i\ \textit{iand next G-next}\ !\ i)$
using *big-ior-map2-defs*[*of* (*map* ($\lambda x.\ x$ *iand F-e*) *G-now*) *G-next* σ ($\lambda x\ y.\ x$ *iand next* y)]
using *28* **by** *fastforce*
have *43*: $(\exists\, i{<}\textit{length}\ (\textit{map}\ (\lambda x.\ x\ \textit{iand F-e})\ \textit{G-now}).\ \sigma \models \textit{map}\ (\lambda x.\ x\ \textit{iand F-e})\ \textit{G-now}\ !\ i\ \textit{iand next G-next}\ !\ i) \Longrightarrow \textit{?lhs}$

```
      proof −
       assume e: (∃ i<length (map (λx. x iand F-e) G-now). σ ⊨ map (λx. x iand F-e) G-now ! i iand
next G-next ! i)
      obtain i where 44: i<length (map (λx. x iand F-e) G-now) ∧ (σ ⊨ map (λx. x iand F-e) G-now !
i iand next G-next ! i)
        using e by blast
       have 45: (ntaken 0 σ) ⊨ ( G-now ! i) iand F-e
        using assms
        by (metis 28 44  iand-defs ntaken-0 nth-map state-qpitl-defs state-qpitl-list-defs)
       have 46: (ntaken 0 σ) ⊨ F
        using 45 assms(1) by fastforce
       have 47: (ndropn 0 σ) ⊨ G-now ! i iand next G-next ! i
        using 44 by fastforce
       have 48: ((ndropn 0 σ) ⊨ (big-ior (map2 (λ x y. x iand (next y))  G-now G-next))) ⟷
                (∃ i<length G-now. ndropn 0 σ ⊨ G-now ! i iand next G-next ! i)
        using big-ior-map2-defs[of G-now G-next (ndropn 0 σ)  (λ x y. x iand (next y))]
        using assms(8) by fastforce
       have 49: (ndropn 0 σ) ⊨ G
        using 44 47 48 assms(5) by auto
       show ?thesis
       by (metis 46 49 semantics-qpitl.simps(5) zero-enat-def zero-le)
      qed
     show ?thesis
     using 29 30 31 42 43 by fastforce
    qed
  qed
qed


lemma GNF-schopstar-step:
 assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))
         state-qpitl F-e
         state-qpitl-list F-now
         length F-now = length F-next
 shows   (σ ⊨ schopstar F) ⟷
          (σ ⊨ (itrue iand empty) ior
               (big-ior (map2 (λ x y. x iand next y) F-now (map (λx. x schop (schopstar F)) F-next))))
        (is ?lhs ⟷ ?rhs)
proof −
 have 1: (σ ⊨ schopstar F) ⟷
         (∃ l. nnth l 0 = 0 ∧ nfinite l ∧ nidx l ∧ enat (nlast l) = nlength σ ∧ nfinite σ ∧
          (∀ i. enat i < nlength l ⟶ (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ F)))
   by simp
 have 2:  ?lhs ⟹ ?rhs
   proof −
    assume l: ?lhs
    obtain l where 3: nnth l 0 = 0 ∧ nfinite l ∧ nidx l ∧ enat (nlast l) = nlength σ ∧ nfinite σ ∧
          (∀ i. enat i < nlength l ⟶ (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ F))
      using l 1 by blast
    have 4:  nlength l = 0 ⟹ ?rhs
```

proof −
assume *n*: *nlength l = 0*
have *5*: *nlength σ = 0*
by (*metis 3 n nnth-nlast the-enat-0 zero-enat-def*)
show *?thesis*
by (*simp add*: *5*)
qed
have *6*: *0< nlength l ⟹ ?rhs*
proof −
assume *g*: *0 < nlength l*
have *7*: *0 < nlength σ*
by (*metis 3 g gr-zeroI less-numeral-extra(3) nfinite-conv-nlength-enat nidx-less-last-1*
nnth-nlast the-enat.simps the-enat-0)
have *8*: (*nsubn σ* (*nnth l 0*) (*nnth l* (*Suc 0*)) ⊨ *F*)
using *3 g* **by** (*metis zero-enat-def*)
have *9*: (*nsubn σ* (*nnth l 0*) (*nnth l* (*Suc 0*))) = (*ntaken* (*nnth l* (*Suc 0*)) *σ*)
using *3* **by** (*simp add*: *nsubn-def1*)
have *10*: (*ntaken* (*nnth l* (*Suc 0*)) *σ*) ⊨ *F*
using *9 8* **by** *auto*
have *11*: (*ntaken* (*nnth l* (*Suc 0*)) *σ*) ⊨ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next*
y)) *F-now F-next*)))
using *10 assms(1) itl-ieq* **by** *blast*
have *12*: (*ntaken* (*nnth l* (*Suc 0*)) *σ*) ⊨ (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))
by (*metis 11 3 7 9 One-nat-def eSuc-enat empty-defs enat-defs(1) g iand-defs ileI1*
ior-defs less-numeral-extra(1) linorder-le-cases nidx-gr-first nsubn-nlength-gr-one
ntaken-all zero-less-iff-neq-zero)
have *13*: *0 < nlength* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)
by (*metis 3 7 9 One-nat-def eSuc-enat enat-defs(1) g ileI1 less-numeral-extra(1)*
linorder-le-cases nidx-gr-first nsubn-nlength-gr-one ntaken-all)
have *14*: (*nlength* (*ntaken* (*nnth l* (*Suc 0*)) *σ*) > *0* ∧
(∃ *i*<*length F-now*. ((*NNil* (*nfirst* (*ntaken* (*nnth l* (*Suc 0*)) *σ*))) ⊨ *F-now* !*i*) ∧
((*ndropn 1* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)) ⊨ *F-next* !*i*)))
using *GNF-defs*[*of F F-e F-now F-next* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)] *12 13*
by (*metis 10 assms(1) assms(2) assms(3) assms(4) order-less-irrefl*)
obtain *i* **where** *15*: *i*<*length F-now* ∧ ((*NNil* (*nfirst* (*ntaken* (*nnth l* (*Suc 0*)) *σ*))) ⊨ *F-now* !*i*) ∧
((*ndropn 1* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)) ⊨ *F-next* !*i*)
using *14* **by** *blast*
have *19*: ((*ndropn 1 σ*) ⊨ *F-next* !*i schop* (*schopstar F*)) ⟷
(∃ *ia*. *enat ia* ≤ *nlength σ* − *enat* (*Suc 0*) ∧
(*ntaken ia* (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*) ∧
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶
(*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*))))
by *simp*
have *191*: *enat* ((*nnth l* (*Suc 0*))−*1*) ≤ *nlength σ* − *enat* (*Suc 0*)
by (*metis 3 One-nat-def eSuc-enat enat-minus-mono1 enat-ord-simps(1) g idiff-enat-enat*
ileI1 nidx-all-le-nlast zero-enat-def)

have *192*: *ntaken* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*
by (*metis 15 191 3 7 One-nat-def Suc-leI diff-add eSuc-enat enat-min-eq g ileI1 less-numeral-extra*(*1*) *nidx-gr-first ntaken-ndropn-swap zero-enat-def*)
have *193*: *nnth* (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *0* = *0*
using *enat-defs*(*1*) **by** *auto*
have *194*: *nfinite* (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
by (*simp add*: *3*)
have *195*: *nidx* (*ndropn 1 l*)
by (*metis 3 g ileI1 nidx-ndropn one-eSuc one-enat-def*)
have *196*: *nnth* (*ndropn 1 l*) *0* = (*nnth l* (*Suc 0*))
by *simp*
have *197*: *nidx* (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
using *nidx-shiftm*[*of* (*ndropn 1 l*) (*nnth l* (*Suc 0*))]
by (*metis 195 196 One-nat-def*)
have *198*: *nlast* (*ndropn* (*Suc 0*) *l*) = *nlast l*
by (*metis Suc-ile-eq g nappend-NNil nappend-ntaken-ndropn nlast-NCons ntaken-0 zero-enat-def*)
have *199*: *nlast* (*nmap* (*λx. x* − *nnth l* (*Suc 0*)) (*ndropn* (*Suc 0*) *l*)) =
nlength σ − *nnth l* (*Suc 0*)
using *nlast-nmap*[*of* (*ndropn* (*Suc 0*) *l*) (*λx. x* − (*nnth l* (*Suc 0*)))]
using *194 nfinite-nmap*
by (*metis 198 3 idiff-enat-enat*)
have *200*: *nlength σ* − *nnth l* (*Suc 0*) = *nlength σ* − *enat* (*Suc 0*) − ((*nnth l* (*Suc 0*))−*1*)
by *auto*
(*metis 3 One-nat-def Suc-pred g ileI1 less-numeral-extra*(*1*) *ndropn-ndropn ndropn-nlength nidx-gr-first one-eSuc one-enat-def plus-1-eq-Suc*)
have *201*: *enat* (*nlast* (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))) =
nlength σ − *enat* (*Suc 0*) − ((*nnth l* (*Suc 0*))−*1*)
using *199 200* **by** *presburger*
have *202*: *nlength* (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) = *nlength l* − (*Suc 0*)
by *force*
have *203*: (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*)) = (*ndropn* (*nnth l* (*Suc 0*)) *σ*)
by (*metis 3 One-nat-def Suc-pred g ileI1 ndropn-ndropn nidx-gr-first one-eSuc one-enat-def plus-1-eq-Suc zero-less-one*)
have *204*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j* =
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
by *simp*
have *2041*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*) =
nnth l ((*Suc j*) + (*Suc 0*)) − (*nnth l* (*Suc 0*))
by (*metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap plus-1-eq-Suc*)
have *205*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)) < *nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
using *204 197* **unfolding** *nidx-expand* **by** *auto*
(*metis 204 One-nat-def add.commute eSuc-enat ileI1 plus-1-eq-Suc*)
have *206*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*))
(*nnth* (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j*)
(*nnth* (*nmap* (*λx. x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*))) =

```
                (nsubn (ndropn (nnth l (Suc 0)) σ)
                     (nnth l (j+ (Suc 0)) −  (nnth l (Suc 0)))
                     (nnth l ((Suc j)+ (Suc 0)) −  (nnth l (Suc 0))))
        using 203 204  2041 by simp
    have 207: ⋀j. j< nlength l − (Suc 0) ⟹
                (nsubn (ndropn (nnth l (Suc 0)) σ)
                     (nnth l (j+ (Suc 0)) −  (nnth l (Suc 0)))
                     (nnth l ((Suc j)+ (Suc 0)) −  (nnth l (Suc 0)))) =
            (nsubn σ (nnth l (Suc j)) (nnth l (Suc (Suc j))))
        using nsubn-ndropn[of - - σ  (nnth l (Suc 0))] by simp
      (metis (no-types, lifting) 197 202 204 205 add.commute add.right-neutral
       bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first order-less-imp-le
        plus-1-eq-Suc zero-less-diff)
    have 208: ⋀j. j< nlength l − (Suc 0) ⟹
                (nsubn σ (nnth l (Suc j)) (nnth l (Suc (Suc j)))) ⊨ F
      using 3
      by (metis eSuc-enat enat-min g ileI1 one-eSuc plus-1-eSuc(2) zero-enat-def)
    have 209: (∀ i. enat i < nlength (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) ⟶
          (nsubn (ndropn ((nnth l (Suc 0))−1 ) (ndropn (Suc 0) σ))
           (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
           (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i)) ⊨ F))
        using 202 206 207 208 by presburger
    have 210: ( enat ((nnth l (Suc 0))−1 ) ≤ nlength σ − enat (Suc 0) ∧
      (ntaken ((nnth l (Suc 0))−1 ) (ndropn (Suc 0) σ) ⊨ F-next ! i) ∧
      ( nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) 0 = 0 ∧
          nfinite (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) ∧
          nidx (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) ∧
          enat (nlast (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l))) =
          nlength σ − enat (Suc 0) − enat ((nnth l (Suc 0))−1 ) ∧ nfinite σ ∧
         (∀ i. enat i < nlength (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) ⟶
           (nsubn (ndropn ((nnth l (Suc 0))−1 ) (ndropn (Suc 0) σ))
                (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
                (nnth (nmap (λx. x − (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i)) ⊨ F))))
       using 191 192 193 194 197 201 209 3  by blast
    have 20: ((ndropn 1 σ) ⊨ F-next !i schop (schopstar F))
      using 210 19 by blast
    have 21: σ ⊨ F-now ! i
      by (metis 15 assms(3) ntaken-nfirst state-qpitl-defs state-qpitl-list-defs)
    have 22: length F-now = length (map (λx.  x schop schopstar F  ) F-next)
      by (simp add: assms(4))
   have 23:  (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx.  x schop schopstar F  ) F-next)))
⟷
          (∃ i<length F-now. σ ⊨ F-now ! i iand next map (λx.  x schop schopstar F  ) F-next ! i)
     using  big-ior-map2-defs[of F-now  (map (λx. x schop (schopstar F)) F-next) σ (λ x y. x iand next
y)]
       22 by auto
    have 24: ( σ ⊨ F-now ! i iand next map (λx.  x schop schopstar F  ) F-next ! i)
      by (metis 15 20 21 7 assms(4) iand-defs ileI1 nth-map one-eSuc semantics-qpitl.simps(4))
    show ?thesis using 15 23 24 ior-defs by blast
  qed
```

show *?thesis*
using *4 6 zero-less-iff-neq-zero* **by** *blast*
qed
have *1000*: *?rhs* $\Longrightarrow$ *?lhs*
proof −
assume *r*: *?rhs*
have *1001*: *nlength* σ = *0* $\vee$ (σ $\models$ *big-ior* (*map2* ($\lambda x\ y.\ x$ *iand next y*) *F-now* (*map* ($\lambda x.$ *x schop schopstar F*) *F-next*)))
using *r* **by** *simp*
have *1002*: *nlength* σ = *0* $\Longrightarrow$ *?lhs*
proof −
assume *n*: *nlength* σ = *0*
have *1003*: *nnth* (*NNil 0*) *0* = *0*
by (*simp add*: *nnth-NNil*)
have *1004*: *nfinite* (*NNil 0*)
by *simp*
have *1005*: *nidx* (*NNil 0*)
using *Suc-ile-eq nidx-expand* **by** *auto*
have *1006*: *enat* (*nlast* (*NNil 0*)) = *nlength* σ
by (*simp add*: *n zero-enat-def*)
have *1007*: *nfinite* σ
using *n*
by (*simp add*: *nfinite-conv-nlength-enat zero-enat-def*)
have *1008*: ($\forall$ *i. enat i* < *nlength* (*NNil 0*) $\longrightarrow$ (*nsubn* σ (*nnth* (*NNil 0*) *i*) (*nnth* (*NNil 0*) (*Suc i*)) $\models$ *F*))
by *auto*
show *?thesis* **apply** *simp*
using *1003 1005 1006 1007 1008* **by** *blast*
qed
have *1009*: (σ $\models$ *big-ior* (*map2* ($\lambda x\ y.\ x$ *iand next y*) *F-now* (*map* ($\lambda x.$ *x schop schopstar F*) *F-next*))) $\Longrightarrow$ *?lhs*
proof −
assume *b*: (σ $\models$ *big-ior* (*map2* ($\lambda x\ y.\ x$ *iand next y*) *F-now* (*map* ($\lambda x.$ *x schop schopstar F*) *F-next*)))
have *1010*: *length F-now* = *length* (*map* ($\lambda x.$ *x schop schopstar F*) *F-next*)
by (*simp add*: *assms*(*4*))
have *1011*: ($\exists$ *i*<*length F-now*. σ $\models$ *F-now* ! *i iand next map* ($\lambda x.$ *x schop schopstar F*) *F-next* ! *i*)
using *1010 b big-ior-map2-defs*[*of F-now* (*map* ($\lambda x.$ *x schop* (*schopstar F*)) *F-next*) σ (λ *x y. x iand next y*)]
by *blast*
obtain *i* **where** *1012*: *i*<*length F-now* $\wedge$ (σ $\models$ *F-now* ! *i iand next map* ($\lambda x.$ *x schop schopstar F*) *F-next* ! *i*)
using *1011* **by** *blast*
have *1013*: (σ $\models$ *F-now* ! *i*)
using *1012* **by** *auto*
have *1014*: ((*ndropn 1* σ) $\models$ *F-next* !*i schop* (*schopstar F*))
using *1012 assms*(*4*) **by** *fastforce*
have *1015*: ((*ndropn 1* σ) $\models$ *F-next* !*i schop* (*schopstar F*)) $\longleftrightarrow$
($\exists$ *ia. enat ia* $\leq$ *nlength* σ − *enat* (*Suc 0*) $\wedge$
(*ntaken ia* (*ndropn* (*Suc 0*) σ) $\models$ *F-next* ! *i*) $\wedge$
($\exists$ *l. nnth l 0* = *0* $\wedge$

nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*))
⊨ *F*))))
by *simp*
obtain *ia* **where** *1016*: *enat ia* ≤ *nlength σ* − *enat* (*Suc 0*) ∧
(*ntaken ia* (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*) ∧
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*))
⊨ *F*)))
using *1014 1015* **by** *blast*
obtain *l* **where** *1017*: *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength σ* − *enat* (*Suc 0*) − *enat ia* ∧ *nfinite σ* ∧
(∀ *i*. *enat i* < *nlength l* ⟶ (*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*))
⊨ *F*))
using *1016* **by** *blast*
have *1018*: *0*<*nlength σ*
using *1012* **by** *force*
have *10183*: *nlength σ* − *enat* (*Suc 0*) − *enat ia* = *nlength σ* − *Suc ia*
by (*metis One-nat-def ndropn-ndropn ndropn-nlength plus-1-eq-Suc*)
have *10184*: *Suc ia* ≤ *nlength σ*
by (*metis 1016 1018 One-nat-def eSuc-enat enat-min-eq ileI1 one-eSuc one-enat-def plus-1-eSuc*(*2*))
have *10185*: *nlength σ* − *Suc ia* + *Suc ia* = *nlength σ*
by (*metis 10184 add.commute enat.simps*(*3*) *enat-add-sub-same le-iff-add*)
have *1019*: (*nlength σ* − *enat* (*Suc 0*) − *enat ia*) + *Suc ia* = *nlength σ*
by (*simp add*: *10183 10185*)
have *1020*: *nlast* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*) = *nlength σ*
by (*metis 1017 1019 nlast-nmap plus-enat-simps*(*1*))
have *1021*: *nlast* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) = *nlength σ*
using *1020* **by** *force*
have *1022*: *nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0* = *0*
by *simp*
have *1023*: *0* < *nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *1*
using *zero-enat-def* **by** *force*
have *1024*: *nfinite* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
by (*simp add*: *1017*)
have *1025*: *nidx* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*))
using *1017 nidx-Ncons-shift* **by** *blast*
have *1028*: (*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+
Suc ia) *l*)) (*Suc 0*))) =
(*nsubn σ 0* (*Suc ia*))
using *1017 zero-enat-def* **by** *fastforce*
have *1029*: ((*nsubn σ 0* (*Suc ia*)) ⊨ (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))) ⟷
(∃ *i*<*length F-now*. *nsubn σ 0* (*Suc ia*) ⊨ *F-now* ! *i iand next F-next* ! *i*)

using *big-ior-map2-defs*[*of F-now F-next* (*nsubn* σ *0* (*Suc ia*)) (λ *x y. x iand* (*next y*))]
assms **by** *blast*
have *1030*: *nsubn* σ *0* (*Suc ia*) $\models$ *F-now* ! *i*
using *1016*
by (*metis 1012 1013 assms*(*3*) *ndropn-0 nsubn-def1 ntaken-nfirst state-qpitl-defs state-qpitl-list-defs*)
have *1031*: *nsubn* σ *0* (*Suc ia*) $\models$ *next F-next* ! *i*
using *1016 ndropn-nsubn*[*of Suc ia* σ *Suc 0 0*]
by (*metis 1019 One-nat-def add-diff-cancel-left*$'$ *enat-le-plus-same*(*2*) *ileI1 le-add1*
nsubn-def1 nsubn-nlength-gr-one one-eSuc plus-1-eq-Suc semantics-qpitl.simps(*4*) *zero-less-Suc*)
have *1034*: (*nsubn* σ (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) (*Suc 0*)) $\models$
((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y. x iand* (*next y*)) *F-now F-next*))))
using *1012 1028 1029 1030 1031* **by** *auto*
have *1035*: $\bigwedge$*j*. *j*<*nlength l* $\Longrightarrow$
(*nnth* ((*nmap* (λ*x. x*+ *Suc ia*) *l*)) *j*) = *nnth l j* + *Suc ia*
by *simp*
have *1036*: $\bigwedge$*j*. *j*<*nlength l* $\Longrightarrow$
(*nnth* ((*nmap* (λ*x. x*+ *Suc ia*) *l*)) (*Suc j*)) = *nnth l* (*Suc j*) + *Suc ia*
using *Suc-ile-eq* **by** *force*
have *1037*: $\bigwedge$*j*. *j*<*nlength l* $\Longrightarrow$
nnth l j + *Suc ia* < *nnth l* (*Suc j*) + *Suc ia*
by (*metis 1017 add-strict-right-mono eSuc-enat ileI1 nidx-expand*)
have *1038*: $\bigwedge$*j*. *j*<*nlength l* $\Longrightarrow$
enat (*nnth l* (*Suc j*) + *Suc ia*) $\leq$ *nlength* σ
using *1017 nidx-expand* **by** *simp*
(*metis 1020 1025 1036 add-Suc-right eSuc-enat enat-ord-simps*(*1*) *ileI1 nfinite-nmap*
nidx-LCons-1 nidx-all-le-nlast nlength-nmap)
have *1039*: $\bigwedge$*j*. *j*<*nlength l* $\Longrightarrow$
Suc 0 + (*nnth l j* + *Suc ia*) $\leq$ *nnth l* (*Suc j*) + *Suc ia*
using *1037* **by** *fastforce*
have *1040*: $\bigwedge$*j*. *j*<*nlength l* $\Longrightarrow$
ndropn (*Suc 0*) (*nsubn* σ (*nnth l j* + *Suc ia*) (*nnth l* (*Suc j*) + *Suc ia*)) =
nsubn σ (*Suc 0* + (*nnth l j* + *Suc ia*)) (*nnth l* (*Suc j*) + *Suc ia*)
using *ndropn-nsubn*[*of* - σ *Suc 0*] **using** *1038 1039* **by** *presburger*
have *1041*: $\bigwedge$*j*. *j*<*nlength l* $\Longrightarrow$
(*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) σ)) (*nnth l j*) (*nnth l* (*Suc j*))) =
nsubn σ ((*nnth l j*) + *Suc ia*) ((*nnth l* (*Suc j*)) + *Suc ia*)
using *nsubn-ndropn*[*of* - - σ *Suc ia*]
by (*metis 1037 One-nat-def add-less-cancel-right ndropn-ndropn plus-1-eq-Suc*)
have *1280*: (*nsubn* σ (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) (*Suc 0*)) $\models$ *F*)
using *1034 assms*(*1*) **by** *simp*
have *1281*: ($\forall$ *i. enat i* < *nlength* ((*nmap* (λ*x. x*+ *Suc ia*) *l*)) $\longrightarrow$
(*nsubn* σ (*nnth* ((*nmap* (λ*x. x*+ *Suc ia*) *l*)) *i*) (*nnth* ((*nmap* (λ*x. x*+ *Suc ia*) *l*)) (*Suc i*)) $\models$ *F*))
using *1017 1036 1041* **by** *auto*
have *1290*: ($\forall$ *i. enat i* < *nlength* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) $\longrightarrow$
(*nsubn* σ (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) (*Suc i*)) $\models$ *F*)) $\longleftrightarrow$
(*nsubn* σ (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x. x*+ *Suc*

ia) l)) (Suc 0)) ⊨ F) ∧
(∀ i. enat i < nlength ((nmap (λx. x+ Suc ia) l)) ⟶
(nsubn σ (nnth ((nmap (λx. x+ Suc ia) l)) i) (nnth ((nmap (λx. x+ Suc ia) l)) (Suc i)) ⊨ F))
using *forall-ncons-split*[*of* (*nmap* (λx. x+ Suc ia) l)
λ x y. (*nsubn σ x y ⊨ F*)] **by** (*simp add*: *1017*)
have *1300*: (∀ i. *enat i < nlength* (*NCons 0* (*nmap* (λx. x+ Suc ia) l)) ⟶
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λx. x+ Suc ia) l)) i) (*nnth* (*NCons 0* (*nmap* (λx. x+ Suc ia) l)) (*Suc i*)) ⊨ *F*))
using *1280 1281 1290* **by** *blast*
have *1400*: ∃ l. *nnth l 0 = 0 ∧ nfinite l ∧ nidx l ∧ enat* (*nlast l*) = *nlength σ ∧ nfinite σ ∧*
(∀ i. *enat i < nlength l* ⟶ (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*))
by (*meson 1017 1021 1022 1025 1300 nfinite-NConsI nfinite-nmap*)
show *?thesis* **using** *1400* **by** *simp*
qed
show *?thesis*
using *1001 1002 1009* **by** *blast*
qed
show *?thesis*
using *1000 2* **by** *blast*
qed

lemma *GNF-omega-step*:
assumes ⊢ *F ieqv* ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ x y. x *iand* (*next y*)) *F-now F-next*)))
state-qpitl F-e
state-qpitl-list F-now
length F-now = length F-next
shows (*σ* ⊨ *omega F*) ⟷
(*σ* ⊨ (*ifalse iand empty*) *ior*
(*big-ior* (*map2* (λ x y. x *iand next y*) *F-now* (*map* (λx. x *schop* (*omega F*)) *F-next*))))
(**is** *?lhs* ⟷ *?rhs*)
proof −
have *1*: (*σ* ⊨ *omega F*) ⟷
(∃ l. ¬*nfinite l ∧ nnth l 0 = 0 ∧ nidx l ∧*
(∀ i. (*nnth l i*) ≤ *nlength σ*) ∧
(∀ i. (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*)))
by *simp*
have *2*: *?lhs* ⟹ *?rhs*
proof −
assume *l*: *?lhs*
obtain *l* **where** *3*: ¬*nfinite l ∧ nnth l 0 = 0 ∧ nidx l ∧*
(∀ i. (*nnth l i*) ≤ *nlength σ*) ∧
(∀ i. (*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*))
using *l 1* **by** *blast*
have *6*: ¬*nfinite σ*
using *3 infinite-nidx-imp-infinite-interval* **by** *blast*
have *7*: *0 < nlength σ*
using *6 enat-defs*(*1*) *nfinite-conv-nlength-enat* **by** *fastforce*

have *8*: (*nsubn σ* (*nnth l 0*) (*nnth l* (*Suc 0*)) ⊨ *F*)
using *3* **by** *blast*
have *9*: (*nsubn σ* (*nnth l 0*) (*nnth l* (*Suc 0*))) = (*ntaken* (*nnth l* (*Suc 0*)) *σ*)
using *3* **by** (*simp add*: *nsubn-def1*)
have *10*: (*ntaken* (*nnth l* (*Suc 0*)) *σ*) ⊨ *F*
using *9 8* **by** *auto*
have *11*: (*ntaken* (*nnth l* (*Suc 0*)) *σ*) ⊨ ((*F-e iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*)))
using *10 assms*(*1*) *itl-ieq* **by** *blast*
have *12*: (*ntaken* (*nnth l* (*Suc 0*)) *σ*) ⊨ (*big-ior* (*map2* (λ *x y*. *x iand* (*next y*)) *F-now F-next*))
by (*metis 11 3 9 One-nat-def empty-defs enat-defs*(*2*) *iand-defs iless-eSuc0 ior-defs less-numeral-extra*(*1*) *less-numeral-extra*(*3*) *nidx-gr-first nlength-eq-enat-nfiniteD not-less nsubn-nlength-gr-one one-eSuc zero-enat-def*)
have *13*: *0* < *nlength* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)
by (*metis 3 9 One-nat-def iless-eSuc0 less-numeral-extra*(*1*) *linorder-not-less nidx-gr-first nlength-eq-enat-nfiniteD nsubn-nlength-gr-one one-eSuc one-enat-def zero-enat-def*)
have *14*: (*nlength* (*ntaken* (*nnth l* (*Suc 0*)) *σ*) > *0* ∧
(∃ *i*<*length F-now*. ((*NNil* (*nfirst* (*ntaken* (*nnth l* (*Suc 0*)) *σ*))) ⊨ *F-now* !*i*) ∧
((*ndropn 1* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)) ⊨ *F-next* !*i*)))
using *GNF-defs*[*of F F-e F-now F-next* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)] *12 13*
by (*metis 10 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *order-less-irrefl*)
obtain *i* **where** *15*: *i*<*length F-now* ∧ ((*NNil* (*nfirst* (*ntaken* (*nnth l* (*Suc 0*)) *σ*))) ⊨ *F-now* !*i*) ∧
((*ndropn 1* (*ntaken* (*nnth l* (*Suc 0*)) *σ*)) ⊨ *F-next* !*i*)
using *14* **by** *blast*
have *19*: ((*ndropn 1 σ*) ⊨ *F-next* !*i schop* (*omega F*)) ⟷
(∃ *ia*. *enat ia* ≤ *nlength σ* − *enat* (*Suc 0*) ∧
(*ntaken ia* (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*) ∧
(∃ *l*. ¬ *nfinite l* ∧
nnth l 0 = *0* ∧ *nidx l* ∧ (∀ *i*. *enat* (*nnth l i*) ≤ *nlength σ* − *enat* (*Suc 0*) − *enat ia*) ∧
(∀ *i*. *nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l i*) (*nnth l* (*Suc i*)) ⊨ *F*)))
by *simp*
have *191*: *enat* ((*nnth l* (*Suc 0*))−*1*) ≤ *nlength σ* − *enat* (*Suc 0*)
by (*metis 3 One-nat-def enat-minus-mono1 idiff-enat-enat*)
have *192*: *ntaken* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) *σ*) ⊨ *F-next* ! *i*
using *13 15 191 3 7*
by (*simp add*: *enat-0-iff*(*1*) *ntaken-ndropn-swap*)
have *193*: *nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *0* = *0*
using *enat-defs*(*1*) **by** *auto*
have *194*: ¬*nfinite* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
by (*simp add*: *3*)
have *195*: *nidx* (*ndropn 1 l*)
by (*metis 3 One-nat-def Suc-ile-eq dual-order.order-iff-strict enat-defs*(*1*) *nfinite-conv-nlength-enat nidx-ndropn zero-le*)
have *196*: *nnth* (*ndropn 1 l*) *0* = (*nnth l* (*Suc 0*))
by *simp*
have *197*: *nidx* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
using *nidx-shiftm*[*of* (*ndropn 1 l*) (*nnth l* (*Suc 0*))]
by (*metis 195 196 One-nat-def*)
have *198*: (∀ *i*. *enat* (*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *i*) ≤ *nlength σ* − *enat* (*Suc 0*) − *enat* ((*nnth l* (*Suc 0*))−*1*))

by (*metis 6 ndropn-nlength nfinite-ndropn nfinite-ntaken nle-le ntaken-all*)
have *202*: *nlength* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) = *nlength l* − (*Suc 0*)
by *force*
have *203*: (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) σ)) = (*ndropn* (*nnth l* (*Suc 0*)) σ)
by (*metis 3 One-nat-def Suc-pred ileI1 less-numeral-extra*(*1*) *ndropn-ndropn nidx-gr-first nlength-eq-enat-nfiniteD one-eSuc one-enat-def plus-1-eq-Suc zero-enat-def zero-less-iff-neq-zero*)
have *204*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j* =
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
by *simp*
have *2041*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*) =
nnth l ((*Suc j*) + (*Suc 0*)) − (*nnth l* (*Suc 0*))
by (*metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap plus-1-eq-Suc*)
have *205*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
nnth l (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)) < *nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*))
using *204 197* **unfolding** *nidx-expand* **apply** *auto*
by (*metis 204 One-nat-def add.commute eSuc-enat ileI1 plus-1-eq-Suc*)
have *206*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) σ))
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *j*)
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc j*))) =
(*nsubn* (*ndropn* (*nnth l* (*Suc 0*)) σ)
(*nnth l* (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))
(*nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*))))
using *203 204 2041* **by** *simp*
have *207*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* (*ndropn* (*nnth l* (*Suc 0*)) σ)
(*nnth l* (*j*+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))
(*nnth l* ((*Suc j*)+ (*Suc 0*)) − (*nnth l* (*Suc 0*)))) =
(*nsubn* σ (*nnth l* (*Suc j*)) (*nnth l* (*Suc* (*Suc j*))))
using *nsubn-ndropn*[*of - - σ* (*nnth l* (*Suc 0*))] **by** *simp*
(*metis* (*no-types*, *lifting*) *197 202 204 205 add.commute add.right-neutral bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first order-less-imp-le plus-1-eq-Suc zero-less-diff*)
have *208*: ⋀*j*. *j*< *nlength l* − (*Suc 0*) ⟹
(*nsubn* σ (*nnth l* (*Suc j*)) (*nnth l* (*Suc* (*Suc j*)))) ⊨ *F*
using *3* **by** *blast*
have *209*: (∀ *i*. *enat i* < *nlength* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) ⟶
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) σ))
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *i*)
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc i*)) ⊨ *F*))
using *202 206 207 208* **by** *presburger*
have *2090*: *nlength* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) = ∞
by (*metis 194 enat2-cases nlength-eq-enat-nfiniteD*)
have *2091*: (∀ *i*.
(*nsubn* (*ndropn* ((*nnth l* (*Suc 0*))−*1*) (*ndropn* (*Suc 0*) σ))
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) *i*)
(*nnth* (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*)) (*Suc i*)) ⊨ *F*))

using *2090 209*
using *enat-ord-code(4)* **by** *presburger*
let *?ia* = ((*nnth l* (*Suc 0*))−*1*)
let *?l* = (*nmap* (λ*x*. *x* − (*nnth l* (*Suc 0*))) (*ndropn* (*Suc 0*) *l*))
have *210*: (*enat ?ia* ≤ *nlength* σ − *enat* (*Suc 0*) ∧
(*ntaken ?ia* (*ndropn* (*Suc 0*) σ) ⊨ *F-next* ! *i*) ∧
(¬*nfinite ?l* ∧
nnth ?l 0 = *0* ∧
nidx ?l ∧
(∀ *i*. *enat* (*nnth ?l i*) ≤ *nlength* σ − *enat* (*Suc 0*) − *enat ?ia*) ∧
(∀ *i*.
(*nsubn* (*ndropn ?ia* (*ndropn* (*Suc 0*) σ))
(*nnth ?l i*)
(*nnth ?l* (*Suc i*)) ⊨ *F*))
)
)
using *191 192 193 194 197 198 2091* **by** *blast*
have *20*: ((*ndropn 1* σ) ⊨ *F-next* !*i schop* (*omega F*))
using *210 19* **by** *blast*
have *21*: σ ⊨ *F-now* ! *i*
by (*metis 15 assms(3) ntaken-nfirst state-qpitl-defs state-qpitl-list-defs*)
have *22*: *length F-now* = *length* (*map* (λ*x*. *x schop omega F*) *F-next*)
by (*simp add*: *assms(4)*)
have *23*: (σ ⊨ *big-ior* (*map2* (λ*x y*. *x iand next y*) *F-now* (*map* (λ*x*. *x schop omega F*) *F-next*))) ⟷
(∃ *i*<*length F-now*. σ ⊨ *F-now* ! *i iand next map* (λ*x*. *x schop omega F*) *F-next* ! *i*)
using *big-ior-map2-defs*[*of F-now* (*map* (λ*x*. *x schop* (*omega F*)) *F-next*) σ (λ *x y*. *x iand next y*)]

22 **by** *auto*
have *24*: (σ ⊨ *F-now* ! *i iand next map* (λ*x*. *x schop omega F*) *F-next* ! *i*)
by (*metis 15 20 21 7 assms(4) iand-defs ileI1 nth-map one-eSuc semantics-qpitl.simps(4)*)
show *?thesis* **using** *15 23 24 ior-defs* **by** *blast*
qed
have *1000*: *?rhs* ⟹ *?lhs*
proof −
assume *r*: *?rhs*
have *1001*: (σ ⊨ *big-ior* (*map2* (λ*x y*. *x iand next y*) *F-now* (*map* (λ*x*. *x schop omega F*) *F-next*)))
using *r* **by** *simp*
have *1010*: *length F-now* = *length* (*map* (λ*x*. *x schop omega F*) *F-next*)
by (*simp add*: *assms(4)*)
have *1011*: (∃ *i*<*length F-now*. σ ⊨ *F-now* ! *i iand next map* (λ*x*. *x schop omega F*) *F-next* ! *i*)
using *1001 1010 big-ior-map2-defs*[*of F-now* (*map* (λ*x*. *x schop* (*omega F*)) *F-next*) σ (λ *x y*. *x iand next y*)]
by *blast*
obtain *i* **where** *1012*: *i*<*length F-now* ∧ (σ ⊨ *F-now* ! *i iand next map* (λ*x*. *x schop omega F*) *F-next* ! *i*)
using *1011* **by** *blast*
have *1013*: (σ ⊨ *F-now* ! *i*)
using *1012* **by** *auto*
have *1014*: ((*ndropn 1* σ) ⊨ *F-next* !*i schop* (*omega F*))
using *1012 assms(4)* **by** *fastforce*

have *1015*: $((ndropn\ 1\ \sigma) \models F\text{-}next\ !i\ schop\ (omega\ F)) \longleftrightarrow$
$(\exists\, ia.\ enat\ ia \leq nlength\ \sigma - enat\ (Suc\ 0) \wedge$
$(ntaken\ ia\ (ndropn\ (Suc\ 0)\ \sigma) \models F\text{-}next\ !\ i) \wedge$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge nidx\ l \wedge$
$(\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ (Suc\ 0) - enat\ ia) \wedge$
$(\forall\, i.\ nsubn\ (ndropn\ ia\ (ndropn\ (Suc\ 0)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models F)))$
by *simp*
obtain *ia* **where** *1016*: $enat\ ia \leq nlength\ \sigma - enat\ (Suc\ 0) \wedge$
$(ntaken\ ia\ (ndropn\ (Suc\ 0)\ \sigma) \models F\text{-}next\ !\ i) \wedge$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge nidx\ l \wedge$
$(\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ (Suc\ 0) - enat\ ia) \wedge$
$(\forall\, i.\ nsubn\ (ndropn\ ia\ (ndropn\ (Suc\ 0)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models F))$
using *1014 1015* **by** *blast*
obtain *l* **where** *1017*: $\neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge nidx\ l \wedge$
$(\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ (Suc\ 0) - enat\ ia) \wedge$
$(\forall\, i.\ nsubn\ (ndropn\ ia\ (ndropn\ (Suc\ 0)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models F)$
using *1016* **by** *blast*
have *1018*: $\neg\ nfinite\ \sigma$
by (*metis 1016 infinite-nidx-imp-infinite-interval ndropn-nlength nfinite-ndropn-a*)
have *1022*: $nnth\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))\ 0 = 0$
by *simp*
have *1023*: $0 < nnth\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))\ 1$
using *zero-enat-def* **by** *force*
have *1024*: $\neg nfinite\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))$
by (*simp add*: *1017*)
have *1025*: $nidx\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))$
using *1017 nidx-Ncons-shift-alt* **by** *blast*
have *1026*: $(\forall\, i.\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))\ i) \leq nlength\ \sigma)$
by (*meson 1018 enat-ile nle-le nlength-eq-enat-nfiniteD*)
have *1028*: $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))\ 0)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))\ (Suc\ 0))) =$
$(nsubn\ \sigma\ 0\ (Suc\ ia))$
using *1017 zero-enat-def* **by** *fastforce*
have *1029*: $((nsubn\ \sigma\ 0\ (Suc\ ia)) \models (big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ \ F\text{-}now\ F\text{-}next))) \longleftrightarrow$
$(\exists\, i{<}length\ F\text{-}now.\ nsubn\ \sigma\ 0\ (Suc\ ia) \models F\text{-}now\ !\ i\ iand\ next\ F\text{-}next\ !\ i)$
using *big-ior-map2-defs*[*of F-now F-next* $(nsubn\ \sigma\ 0\ (Suc\ ia))\ (\lambda\ x\ y.\ x\ iand\ (next\ y))$]
assms **by** *blast*
have *1030*: $nsubn\ \sigma\ 0\ (Suc\ ia) \models F\text{-}now\ !\ i$
using *1016*
by (*metis 1012 1013 assms*(*3*) *ndropn-0 nsubn-def1 ntaken-nfirst state-qpitl-defs state-qpitl-list-defs*)
have *1031*: $nsubn\ \sigma\ 0\ (Suc\ ia) \models\ \ next\ F\text{-}next\ !\ i$
using *1016 ndropn-nsubn*[*of Suc ia* σ *Suc 0 0*]
by (*metis 1022 1023 1026 1028 One-nat-def add.commute diff-zero enat-min-eq enat-ord-simps*(*1*)
ileI1 leD ndropn-0 nsubn-def1 nsubn-nlength-gr-one ntaken-ndropn-swap one-eSuc one-enat-def
plus-1-eq-Suc semantics-qpitl.simps(*4*) *zero-less-iff-neq-zero*)
have *1034*: $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))\ 0)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x.\ x+\ Suc\ ia)\ l))\ (Suc\ 0)) \models$

((F-e iand empty) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next))))
using *1012 1028 1029 1030 1031* **by** *auto*
have *1035*: ⋀*j*. *j*<*nlength l* ⟹
(*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *j*) = *nnth l j* + *Suc ia*
by *simp*
have *1036*: ⋀*j*. *j*<*nlength l* ⟹
(*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc j*)) = *nnth l* (*Suc j*) + *Suc ia*
using *Suc-ile-eq* **by** *force*
have *1037*: ⋀*j*. *j*<*nlength l* ⟹
nnth l j + *Suc ia* < *nnth l* (*Suc j*) + *Suc ia*
by (*metis 1017 add-strict-right-mono eSuc-enat ileI1 nidx-expand*)
have *1038*: ⋀*j*. *j*<*nlength l* ⟹
enat (*nnth l* (*Suc j*) + *Suc ia*) ≤ *nlength σ*
using *1017 nidx-expand* **by** *simp*
(*meson 1018 enat-ile nle-le nlength-eq-enat-nfiniteD*)
have *1039*: ⋀*j*. *j*<*nlength l* ⟹
Suc 0 + (*nnth l j* + *Suc ia*) ≤ *nnth l* (*Suc j*) + *Suc ia*
using *1037* **by** *fastforce*
have *1040*: ⋀*j*. *j*<*nlength l* ⟹
ndropn (*Suc 0*) (*nsubn σ* (*nnth l j* + *Suc ia*) (*nnth l* (*Suc j*) + *Suc ia*)) =
nsubn σ (*Suc 0* + (*nnth l j* + *Suc ia*)) (*nnth l* (*Suc j*) + *Suc ia*)
using *ndropn-nsubn*[*of - σ Suc 0*] **using** *1038 1039* **by** *presburger*
have *1041*: ⋀*j*. *j*<*nlength l* ⟹
(*nsubn* (*ndropn ia* (*ndropn* (*Suc 0*) *σ*)) (*nnth l j*) (*nnth l* (*Suc j*))) =
nsubn σ ((*nnth l j*) + *Suc ia*) ((*nnth l* (*Suc j*)) + *Suc ia*)
using *nsubn-ndropn*[*of - - σ Suc ia*]
by (*metis 1037 One-nat-def add-less-cancel-right ndropn-ndropn plus-1-eq-Suc*)
have *1280*: (*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*)) ⊨ *F*)
using *1034 assms*(*1*) **by** *simp*
have *1281*: (∀ *i*. *enat i* < *nlength* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *1017 1036 1041*
by (*metis 1035 nlength-nmap*)
have *1290*: (∀ *i*. *enat i* < *nlength* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*)) ⟷
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *0*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc 0*)) ⊨ *F*) ∧
(∀ *i*. *enat i* < *nlength* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* ((*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *forall-ncons-split-alt*[*of* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)
λ *x y*. (*nsubn σ x y* ⊨ *F*)]
by (*simp add*: *1017*)
have *1300*: (∀ *i*. *enat i* < *nlength* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) ⟶
(*nsubn σ* (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) *i*) (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *Suc ia*) *l*)) (*Suc i*)) ⊨ *F*))
using *1280 1281 1290* **by** *blast*

```
  have 1340: nlength (NCons 0 (nmap (λx. x+ Suc ia) l)) = ∞
    by (metis 1024 enat2-cases nlength-eq-enat-nfiniteD)
  have 1350: (∀ i.
        (nsubn σ (nnth (NCons 0 (nmap (λx. x+ Suc ia) l)) i) (nnth (NCons 0 (nmap (λx. x+ Suc ia) l))
(Suc i)) ⊨ F))
      using 1300 1340 enat-ord-code(4) by presburger
  have 1400: (∃(l:: nat nellist).
            ¬nfinite l ∧ (nnth l 0) = 0 ∧ nidx l ∧
            (∀ i. (nnth l i) ≤ nlength σ) ∧
            (∀ i. ( (nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ F) )
          )
    by (meson 1017 1022 1025 1026 1350 nfinite-NCons nfinite-nmap)
     show ?thesis using 1400 by simp
    qed
 show ?thesis
 using 1000 2 by blast
qed


lemma big-ior-dist-exists:
 ⊢ (Ex n. big-ior L) ieqv big-ior (map (λ x. (Ex n. x)) L)
apply (auto simp add: big-ior-defs)
apply fastforce
by blast

lemma more-defs [simp]:
 (σ ⊨ more) = ( (nlength σ) > 0)
by (auto simp add: empty-d-def more-d-def ileI1 one-eSuc)


lemma state-qpitl-exists-dist-iand:
 assumes state-qpitl w
 shows ⊢ ((Ex n. w iand next f) ieqv (Ex n. w) iand next (Ex n. f))
proof −
 have 1: ⋀ σ.
        (σ ⊨ (Ex n. w iand next f)) ⟹ (σ ⊨ (Ex n. w) iand next (Ex n. f))
 using assms by (auto simp add: nlength-upd)
  (metis One-nat-def ndropn-nlength ndropn-upd one-enat-def)
 have 2: ⋀ σ. (σ ⊨ (Ex n. w) iand next (Ex n. f)) ⟹ (σ ⊨ (Ex n. w iand next f))
  proof −
   fix σ
   assume a: (σ ⊨ (Ex n. w) iand next (Ex n. f))
   show (σ ⊨ (Ex n. w iand next f))
   proof −
    have 3: (σ ⊨ (Ex n. w))
      using a by auto
    obtain l where 4: nlength l = nlength σ ∧ (upd σ n l ⊨ w)
      using 3 by auto
    have 5: nlength σ > 0 ∧(ndropn 1 σ ⊨ (Ex n. f))
     using a by auto
```

obtain *l1* **where** *6*: *nlength l1 = nlength(ndropn 1 σ) ∧ (upd (ndropn 1 σ) n l1 ⊨ f)*
using *5* **by** *auto*
have *7*: *(NNil (nfirst (upd σ n l)) ⊨ w)*
using *assms 4 state-qpitl-defs* **by** *blast*
have *8*: *(NNil (nfirst (upd σ n (nappend (NNil (nfirst l)) l1)))) =*
(NNil (nfirst (upd σ n l)))
by *(metis 4 5 6 One-nat-def nappend-NNil ndropn-0 ndropn-Suc-conv-ndropn nfirst-eq-nnth-zero nfirst-upd nlength-NCons nnth-0 zero-enat-def)*
have *9*: *(NNil (nfirst (upd σ n (nappend (NNil (nfirst l)) l1)))) ⊨ w*
using *7 8* **by** *fastforce*
have *10*: *(ndropn 1 (upd σ n (nappend (NNil (nfirst l)) l1)) ⊨ f)*
by *(metis 5 6 One-nat-def ileI1 nappend-NNil ndropn-0 ndropn-Suc-NCons ndropn-Suc-conv-ndropn ndropn-upd nlength-NCons one-eSuc one-enat-def zero-enat-def)*
have *11*: *nlength (nappend (NNil (nfirst l)) l1) = nlength σ*
by *(metis 5 6 One-nat-def nappend-NNil ndropn-0 ndropn-Suc-conv-ndropn nlength-NCons zero-enat-def)*
have *12*: *(upd σ n (nappend (NNil (nfirst l)) l1) ⊨ w iand next f)*
by *(metis 10 11 9 a assms iand-defs nlength-upd semantics-qpitl.simps(4) state-qpitl-defs)*
have *13*: *(σ ⊨ (Ex n. w iand next f)) ⟷*
(∃ l. nlength l = nlength σ ∧ (upd σ n l ⊨ w iand next f))
by *(metis semantics-qpitl.simps(8))*
show *?thesis*
using *11 12 13* **by** *blast*
qed
qed
show *?thesis* **unfolding** *valid-def*
by *(metis 1 2 ieqv-defs)*
qed

lemma *big-ior-state-qpitl-exists-dist-iand*:
assumes *state-qpitl-list F-now*
length F-now = length F-next
shows *⊢ ((Ex n. (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))*
ieqv
(big-ior (map2 (λ x y. x iand (next y))
(map (λx. (Ex n. x)) F-now)
(map (λx. (Ex n. x)) F-next))))
proof −
have *1*: *⊢ (Ex n. (big-ior (map2 (λ x y. x iand (next y)) F-now F-next))) ieqv*
(big-ior (map (λ x. (Ex n. x)) (map2 (λ x y. x iand (next y)) F-now F-next)))
using *big-ior-dist-exists* **by** *auto*
have *2*: *⋀σ. (σ ⊨ ((big-ior (map (λ x. (Ex n. x)) (map2 (λ x y. x iand (next y)) F-now F-next))))) =*
(σ ⊨ (big-ior (map2 (λ x y. x iand (next y))
(map (λx. (Ex n. x)) F-now)
(map (λx. (Ex n. x)) F-next))))
proof −
fix *σ*
show *(σ ⊨ big-ior (map (exists-d n) (map2 (λx y. x iand next y) F-now F-next))) =*
(σ ⊨ big-ior (map2 (λx y. x iand next y) (map (exists-d n) F-now) (map (exists-d n) F-next)))
proof −

```
    have 3: (σ ⊨ big-ior (map (exists-d n) (map2 (λx y. x iand next y) F-now F-next))) =
          (∃ i<length (map (exists-d n) (map2 (λx y. x iand next y) F-now F-next)).
            σ ⊨ map (exists-d n) (map2 (λx y. x iand next y) F-now F-next) ! i)
     unfolding big-ior-defs by simp
    have 4: ⋀ i. i< length F-now ⟹
                (σ ⊨ map (exists-d n) (map2 (λx y. x iand next y) F-now F-next) ! i) =
                (σ ⊨ (Ex n. F-now ! i iand next F-next ! i))
      by (simp add: assms(2))
    have 5: ⋀ i. i< length F-now ⟹
            (σ ⊨ (Ex n. F-now ! i iand next F-next ! i)) =
            (σ ⊨ (Ex n. F-now ! i) iand next (Ex n. F-next ! i))
      using assms(1) itl-ieq state-qpitl-exists-dist-iand state-qpitl-list-defs by blast
    have 6: (σ ⊨ (big-ior (map2 (λ x y. x iand (next y))
                (map (λx. (Ex n. x)) F-now)
                (map (λx. (Ex n. x)) F-next)))) =
          (∃ i<length (map2 (λx y. x iand next y) (map (exists-d n) F-now) (map (exists-d n) F-next)).
            σ ⊨ map2 (λx y. x iand next y) (map (exists-d n) F-now) (map (exists-d n) F-next) ! i)
      unfolding big-ior-defs  by simp
    have 7: ⋀ i . i< length F-now ⟹
            (σ ⊨ map2 (λx y. x iand next y) (map (exists-d n) F-now) (map (exists-d n) F-next) ! i) =
            (σ ⊨ ((map (exists-d n) F-now) ! i) iand next ((map (exists-d n) F-next) ! i))
      by (simp add: assms(2))
    have 8: ⋀ i . i< length F-now ⟹
            (σ ⊨ ((map (exists-d n) F-now) ! i) iand next ((map (exists-d n) F-next) ! i)) =
            (σ ⊨ (Ex n. F-now ! i) iand next (Ex n. F-next ! i))
      using assms(2) by auto
    show ?thesis
    using 3 5 6 assms(2) by auto
   qed
  qed
 show ?thesis
 using 1 2 by auto
qed



lemma GNF-exists-step:
 assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))
         state-qpitl F-e
         state-qpitl-list F-now
         length F-now = length F-next
 shows   (σ ⊨ (Ex n. F) ) ⟷
          (σ ⊨ ((Ex n. F-e) iand empty) ior
               (big-ior (map2 (λ x y. x iand next y)
                        (map (λx. (Ex n. x)) F-now)
                        (map (λx. (Ex n. x)) F-next))))
proof −
  have 1: (σ ⊨ (Ex n. F) ) =
          (σ ⊨ (Ex n. (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))  F-now F-next)))))
   using assms by simp
```

have *2*: $(\sigma \models (Ex\ n.\ ((\ F\text{-}e\ iand\ empty\ )\ ior\ (big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ \ F\text{-}now\ F\text{-}next)))))$
=
$((\sigma \models (Ex\ n.\ (F\text{-}e\ iand\ empty))) \vee (\sigma \models (Ex\ n.\ (big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ \ F\text{-}now\ F\text{-}next)))))$
by *auto*
have *3*: $(\sigma \models (Ex\ n.\ (F\text{-}e\ iand\ empty))) = (\sigma \models (Ex\ n.\ F\text{-}e)\ iand\ empty)$
using *nlength-upd* **by** *auto*
have *4*: $(\sigma \models (Ex\ n.\ (big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ \ F\text{-}now\ F\text{-}next)))) =$
$(\sigma \models (big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ next\ y)$
$(map\ (\lambda x.\ (Ex\ n.\ x))\ F\text{-}now)$
$(map\ (\lambda x.\ (Ex\ n.\ x))\ F\text{-}next))))$
using *assms big-ior-state-qpitl-exists-dist-iand itl-ieq* **by** *blast*
have *5*: $(\sigma \models ((Ex\ n.\ F\text{-}e)\ iand\ empty)\ ior$
$(big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ next\ y)$
$(map\ (\lambda x.\ (Ex\ n.\ x))\ F\text{-}now)$
$(map\ (\lambda x.\ (Ex\ n.\ x))\ F\text{-}next)))) =$
$((\sigma \models (Ex\ n.\ F\text{-}e)\ iand\ empty) \vee$
$(\sigma \models (big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ next\ y)$
$(map\ (\lambda x.\ (Ex\ n.\ x))\ F\text{-}now)$
$(map\ (\lambda x.\ (Ex\ n.\ x))\ F\text{-}next)))\ )$
by *auto*
show *?thesis*
using *1 2 3 4 5* **by** *blast*
qed

lemma *GNF-atom-base*:
$\vdash (\$n)\ ieqv\ (GNF\ (\$n))$
unfolding *GNF-def GNF-cons-def big-ior-def*
by (*auto*)
(*metis add-0 ileI1 linorder-not-less not-less-zero order-neq-le-trans plus-1-eSuc*(*2*))

lemma *GNF-ifalse-base*:
$\vdash ifalse\ ieqv\ (GNF\ ifalse)$
unfolding *GNF-def GNF-cons-def big-ior-def*
by *auto*

lemma *GNF-next-step*:
$\vdash (next\ f)\ ieqv\ (GNF\ (next\ f))$
unfolding *GNF-def GNF-cons-def big-ior-def*
by (*auto*)

lemma *exists-GNF-nondet*:
$\exists\ ae\ a\ A.\ (state\text{-}qpitl\ ae) \wedge (state\text{-}qpitl\text{-}list\ a) \wedge length\ a = length\ A \wedge$
$(\vdash f\ ieqv\ (ae\ iand\ empty)\ ior\ (big\text{-}ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ next\ y)\ a\ A)))$
proof (*induction f*)
case *false-d*
then show *?case*

```
  proof −
   have f1: ⊢ ifalse ieqv (GNF ifalse)
     using GNF-ifalse-base by auto
   have f2: state-qpitl-list (GNF-state ifalse)
     unfolding state-qpitl-list-def by (simp add: inot-d-def itrue-d-def)
   have f3: length (GNF-state ifalse) = length (GNF-next ifalse)
     by simp
   show ?thesis
   by (metis f1 f2 f3 GNF-cons-def GNF-def GNF-empty.simps(1) state-qpitl.simps(1))
  qed
next
case (atom-d x)
then show ?case
  proof −
   have a1: ⊢($x) ieqv (GNF ($ x))
     using GNF-atom-base by blast
   have a2: state-qpitl-list (GNF-state ($x))
     unfolding state-qpitl-list-def by simp
   have a3: length (GNF-state ($x)) = length (GNF-next ($x))
    by simp
   show ?thesis
   by (metis GNF-cons-def GNF-def GNF-empty.simps(2) a1 a2 a3 state-qpitl.simps(2))
  qed
next
case (iimp-d f1 f2)
then show ?case
 proof −
  obtain ae1 a1 A1 where i1: state-qpitl ae1 ∧ state-qpitl-list a1 ∧ length a1 = length A1 ∧
     (⊢ f1 ieqv ae1 iand empty ior big-ior (map2 (λx y. x iand next y) a1 A1))
     using iimp-d.IH(1) by blast
  obtain ae2 a2 A2 where i2: state-qpitl ae2 ∧ state-qpitl-list a2 ∧ length a2 = length A2 ∧
     (⊢ f2 ieqv ae2 iand empty ior big-ior (map2 (λx y. x iand next y) a2 A2))
     using iimp-d.IH(2) by blast
  have i3: ⊢ (f1 iimp f2) ieqv ( ( (inot ae1) ior ae2) iand empty) ior
            (big-ior
             (map2 (λ x y. x iand (next y))
                  ((add-to-now a1 ) @ a2)
                  ((map (λx. (inot x)) (add-to-next A1 )) @ A2) ))
      using GNF-iimp-step i1 i2 itl-ieq by force
  have i4: length ((add-to-now a1 ) @ a2) = length ((map (λx. (inot x)) (add-to-next A1 )) @ A2)
    by (simp add: add-to-next-def add-to-now-def i1 i2 length-add-to-now-next-gen)
  have i5: state-qpitl-list ((add-to-now a1 ) @ a2)
    by (simp add: add-to-now-def i1 i2 state-qpitl-foldr-add-to-now-single-inv-a state-qpitl-list-append)
   show ?thesis
   by (metis i1 i2 i3 i4 i5 inot-d-def ior-d-def state-qpitl.simps(1) state-qpitl.simps(3))
 qed
next
case (next-d f)
then show ?case
  proof −
```

```
    obtain ae a A where n1:  state-qpitl ae ∧ state-qpitl-list a ∧ length a = length A ∧
      (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
      using next-d.IH by blast
    have n2: ⊢ (next f) ieqv (GNF (next f))
     using GNF-next-step by blast
    have n3: state-qpitl-list (GNF-state (next f))
      by (simp add: inot-d-def itrue-d-def state-qpitl-list-defs)
    have n4: length (GNF-state (next f)) = length (GNF-next (next f))
      by simp
    show ?thesis
    by (metis GNF-cons-def GNF-def GNF-empty.simps(4) n2 n3 n4 state-qpitl.simps(1))
  qed
next
case (chop-d f1 f2)
then show ?case
  proof −
    obtain ae1 a1 A1 where c1: state-qpitl ae1 ∧ state-qpitl-list a1 ∧ length a1 = length A1 ∧
      (⊢ f1 ieqv ae1 iand empty ior big-ior (map2 (λx y. x iand next y) a1 A1) )
     using chop-d.IH(1) by blast
    obtain ae2 a2 A2 where c2: state-qpitl ae2 ∧ state-qpitl-list a2 ∧ length a2 = length A2 ∧
      (⊢ f2 ieqv ae2 iand empty ior big-ior (map2 (λx y. x iand next y) a2 A2) )
     using chop-d.IH(2) by blast
    have c3: ⊢ f1 schop f2 ieqv (ae1 iand ae2) iand empty ior
              (big-ior (map2 (λ x y. x iand next y)
                       (a1 @ (map (λx. x iand ae1) a2))
                       ((map (λx. x schop f2) A1) @  A2)))
      using GNF-schop-step[of f1 ae1 a1 A1 f2 ae2 a2 A2] c1 c2 itl-ieq by blast
    have c4: length (a1 @ (map (λx. x iand ae1) a2)) = length ((map (λx. x schop f2) A1) @  A2)
       by (simp add: c1 c2)
    have c5: state-qpitl-list (a1 @ (map (λx. x iand ae1) a2))
     by (simp add: c1 c2 state-qpitl-list-append state-qpitl-list-map-alt)
    have c6:  state-qpitl (ae1 iand ae2)
        by (simp add: c1 c2 iand-d-def inot-d-def ior-d-def)
    show ?thesis using c1 c2 c3 c4 c5 c6 by blast
  qed
next
case (schopstar-d f)
then show ?case
  proof −
    obtain ae a A where s1:  state-qpitl ae ∧ state-qpitl-list a ∧ length a = length A ∧
      (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
      using schopstar-d.IH by blast
    have s2: ⊢ schopstar f ieqv (itrue iand empty) ior
              (big-ior (map2 (λ x y. x iand next y) a (map (λx. x schop (schopstar f)) A)))
      using GNF-schopstar-step[of f ae a A] s1 using itl-ieq by blast
    have s3: length a = length (map (λx. x schop (schopstar f)) A)
      by (simp add: s1)
    have s4: state-qpitl itrue
       by (simp add: inot-d-def itrue-d-def)
    show ?thesis
```

```
    by (metis  s1 s2 s3 s4)
   qed
next
case (omega-d f)
then show ?case
   proof −
    obtain ae a A where o1:  state-qpitl ae ∧ state-qpitl-list a ∧ length a = length A ∧
      (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
      using omega-d.IH by blast
    have o2: ⊢ omega f ieqv (ifalse iand empty) ior
                (big-ior (map2 (λ x y. x iand next y) a (map (λx. x schop (omega f)) A)))
      using GNF-omega-step[of f ae a A] o1 using itl-ieq by blast
    have o3: length a = length (map (λx. x schop (omega f)) A)
      by (simp add: o1)
    show ?thesis
    using o1 o2 o3 state-qpitl.simps(1) by blast
   qed
next
case (exists-d n f)
then show ?case
  proof −
   obtain ae a A where e1:  state-qpitl ae ∧ state-qpitl-list a ∧ length a = length A ∧
      (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
    using exists-d.IH by blast
   have e2: ⊢ (Ex n. f) ieqv
              ((Ex n. ae) iand empty) ior (big-ior (map2 (λ x y. x iand next y)
                 (map (λ x. (Ex n. x)) a) (map (λ x. (Ex n. x)) A)))
     using GNF-exists-step[of f ae a A - n] e1 using itl-ieq  by blast
   have e3: length (map (λ x. (Ex n. x)) a) = length (map (λ x. (Ex n. x)) A)
      using e1 by simp
   have e4: state-qpitl (Ex n. ae)
     by (simp add: e1)
   have e5: state-qpitl-list (map (λ x. (Ex n. x)) a)
     using e1 state-qpitl-list-defs by auto
   show ?thesis
    using e1 e2 e3 e4 e5 by blast
  qed
qed

lemma state-qpitl-GNF-empty:
 state-qpitl (GNF-empty f)
proof (induction f)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case by simp
next
case (iimp-d f1 f2)
then show ?case  apply simp  unfolding inot-d-def ior-d-def by simp
```

next
case (*next-d f*)
then show *?case* **by** *simp*
next
case (*chop-d f1 f2*)
then show *?case* **apply** *simp* **unfolding** *iand-d-def inot-d-def ior-d-def* **by** *simp*
next
case (*schopstar-d f*)
then show *?case* **apply** *simp* **unfolding** *itrue-d-def inot-d-def* **by** *simp*
next
case (*omega-d f*)
then show *?case* **by** *simp*
next
case (*exists-d x1 f*)
then show *?case* **by** *simp*
qed

lemma *state-qpitl-map-exists*:
assumes *state-qpitl-list xs*
shows *state-qpitl-list* (*map* (*λ x. Ex n. x*) *xs*)
using *assms* **unfolding** *state-qpitl-list-defs* **by** *simp*

lemma *state-qpitl-list-GNF-state*:
state-qpitl-list (*GNF-state f*)
proof (*induction f*)
case *false-d*
then show *?case* **unfolding** *state-qpitl-list-def* **apply** *simp* **unfolding** *itrue-d-def inot-d-def* **by** *simp*
next
case (*atom-d x*)
then show *?case* **unfolding** *state-qpitl-list-def* **by** *simp*
next
case (*iimp-d f1 f2*)
then show *?case* **unfolding** *state-qpitl-list-def* **apply** *simp*
using *add-to-now-def state-qpitl-foldr-add-to-now-single-inv-a state-qpitl-list-def* **by** *auto*
next
case (*next-d f*)
then show *?case* **unfolding** *state-qpitl-list-def* **apply** *simp* **unfolding** *itrue-d-def inot-d-def* **by** *simp*
next
case (*chop-d f1 f2*)
then show *?case* **unfolding** *state-qpitl-list-def* **apply** *simp*
using *state-qpitl-GNF-empty state-qpitl-list-def state-qpitl-list-map-alt* **by** *force*
next
case (*schopstar-d f*)
then show *?case* **unfolding** *state-qpitl-list-def* **by** *simp*
next
case (*omega-d f*)
then show *?case* **unfolding** *state-qpitl-list-def* **by** *simp*
next

case (*exists-d x1 f*)
then show *?case* **unfolding** *state-qpitl-list-def* **apply** *simp*
using *state-qpitl-list-def state-qpitl-map-exists* **by** *blast*
qed

lemma *length-GNF-state-eq-length-GNF-next*:
length (*GNF-state f*) = *length* (*GNF-next f*)
proof (*induction f*)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case* **by** *simp*
next
case (*iimp-d f1 f2*)
then show *?case*
by (*simp add*: *add-to-next-def add-to-now-def length-add-to-now-next-gen*)
next
case (*next-d f*)
then show *?case* **by** *simp*
next
case (*chop-d f1 f2*)
then show *?case* **by** *simp*
next
case (*schopstar-d f*)
then show *?case* **by** *simp*
next
case (*omega-d f*)
then show *?case* **by** *simp*
next
case (*exists-d x1 f*)
then show *?case* **by** *simp*
qed

lemma *ieqv-GNF*:
⊢ *f ieqv GNF f*
proof (*induction f*)
case *false-d*
then show *?case*
using *GNF-ifalse-base* **by** *auto*
next
case (*atom-d x*)
then show *?case* **using** *GNF-atom-base* **by** *blast*
next
case (*iimp-d f1 f2*)
then show *?case*
proof −
have *i1*: ⊢ *f1 ieqv GNF-empty f1 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f1*)

(*GNF-next f1*))
using *iimp-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *i2*: ⊢ *f2 ieqv GNF-empty f2 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f2*) (*GNF-next f2*))
using *iimp-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *i3*: *state-qpitl* (*GNF-empty f1*)
by (*simp add*: *state-qpitl-GNF-empty*)
have *i4*: *state-qpitl-list* (*GNF-state f1*)
by (*simp add*: *state-qpitl-list-GNF-state*)
have *i5*: *state-qpitl* (*GNF-empty f2*)
by (*simp add*: *state-qpitl-GNF-empty*)
have *i6*: *state-qpitl-list* (*GNF-state f2*)
by (*simp add*: *state-qpitl-list-GNF-state*)
have *i7*: *length* (*GNF-state f1*) = *length* (*GNF-next f1*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
have *i8*: *length* (*GNF-state f2*) = *length* (*GNF-next f2*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
show *?thesis* **using**
GNF-iimp-step[*of f1 GNF-empty f1* (*GNF-state f1*) (*GNF-next f1*)
f2 GNF-empty f2 (*GNF-state f2*) (*GNF-next f2*)]
using *GNF-cons-def GNF-def i1 i2 i3 i4 i5 i6 i7 i8* **by** *fastforce*
qed
next
case (*next-d f*)
then show *?case* **using** *GNF-next-step* **by** *blast*
next
case (*chop-d f1 f2*)
then show *?case*
proof −
have *c1*: ⊢ *f1 ieqv GNF-empty f1 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f1*) (*GNF-next f1*))
using *chop-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *c2*: ⊢ *f2 ieqv GNF-empty f2 iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f2*) (*GNF-next f2*))
using *chop-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *c3*: *state-qpitl* (*GNF-empty f1*)
by (*simp add*: *state-qpitl-GNF-empty*)
have *c4*: *state-qpitl-list* (*GNF-state f1*)
by (*simp add*: *state-qpitl-list-GNF-state*)
have *c5*: *state-qpitl* (*GNF-empty f2*)
by (*simp add*: *state-qpitl-GNF-empty*)
have *c6*: *state-qpitl-list* (*GNF-state f2*)
by (*simp add*: *state-qpitl-list-GNF-state*)
have *c7*: *length* (*GNF-state f1*) = *length* (*GNF-next f1*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
have *c8*: *length* (*GNF-state f2*) = *length* (*GNF-next f2*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
show *?thesis* **using**
GNF-schop-step[*of f1 GNF-empty f1* (*GNF-state f1*) (*GNF-next f1*)
f2 GNF-empty f2 (*GNF-state f2*) (*GNF-next f2*)]

unfolding *GNF-def GNF-cons-def* **using** *c1 c2 c3 c4 c5 c6 c7 c8* **by** *fastforce*
qed
next
case (*schopstar-d f*)
then show *?case*
proof −
have *s1*: ⊢ *f ieqv GNF-empty f iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f*) (*GNF-next f*))
using *schopstar-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *s2*: *state-qpitl* (*GNF-empty f*)
by (*simp add*: *state-qpitl-GNF-empty*)
have *s3*: *state-qpitl-list* (*GNF-state f*)
by (*simp add*: *state-qpitl-list-GNF-state*)
have *s4*: *length* (*GNF-state f*) = *length* (*GNF-next f*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
show *?thesis* **using**
GNF-schopstar-step[*of f GNF-empty f* (*GNF-state f*) (*GNF-next f*)]
unfolding *GNF-def GNF-cons-def* **using** *s1 s2 s3 s4* **by** *fastforce*
qed
next
case (*omega-d f*)
then show *?case*
proof −
have *o1*: ⊢ *f ieqv GNF-empty f iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f*) (*GNF-next f*))
using *omega-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *o2*: *state-qpitl* (*GNF-empty f*)
by (*simp add*: *state-qpitl-GNF-empty*)
have *o3*: *state-qpitl-list* (*GNF-state f*)
by (*simp add*: *state-qpitl-list-GNF-state*)
have *o4*: *length* (*GNF-state f*) = *length* (*GNF-next f*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
show *?thesis* **using**
GNF-omega-step[*of f GNF-empty f* (*GNF-state f*) (*GNF-next f*)]
unfolding *GNF-def GNF-cons-def* **using** *o1 o2 o3 o4* **by** *fastforce*
qed
next
case (*exists-d x1 f*)
then show *?case*
proof −
have *e1*: ⊢ *f ieqv GNF-empty f iand empty ior big-ior* (*map2* (λ*x y*. *x iand next y*) (*GNF-state f*) (*GNF-next f*))
using *exists-d* **by** (*simp add*: *GNF-cons-def GNF-def*)
have *e2*: *state-qpitl* (*GNF-empty f*)
by (*simp add*: *state-qpitl-GNF-empty*)
have *e3*: *state-qpitl-list* (*GNF-state f*)
by (*simp add*: *state-qpitl-list-GNF-state*)
have *e4*: *length* (*GNF-state f*) = *length* (*GNF-next f*)
by (*simp add*: *length-GNF-state-eq-length-GNF-next*)
show *?thesis* **using**

GNF-exists-step[*of f GNF-empty f* (*GNF-state f*) (*GNF-next f*)]
unfolding *GNF-def GNF-cons-def* **using** *e1 e2 e3 e4* **by** *fastforce*
qed
qed

lemma *exist-GNF-det*:
GNF-prop f
proof −
have *1*: ∃ *ae a A*. (*state-qpitl ae*) ∧ (*state-qpitl-list a*) ∧ *length a* = *length A* ∧
(⊢ *f ieqv* (*ae iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand next y*) *a A*)))
using *exists-GNF-nondet* **by** *blast*
obtain *ae a A* **where** *2*: (*state-qpitl ae*) ∧ (*state-qpitl-list a*) ∧ *length a* = *length A* ∧
(⊢ *f ieqv* (*ae iand empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand next y*) *a A*)))
using *1* **by** *blast*
have *3*: ⊢ (*big-ior* (*map2* (λ *x y*. *x iand next y*) *a A*)) *ieqv*
big-ior (*map2* (λ *x y*. *x iand* (*next y*))
(*foldr* (*add-to-now-single*) *a* [*itrue*])
(*foldr* (*add-to-next-single*) *A* [*ifalse*]))
using *2 det-big-ior-ieqv-nondet-big-ior-a* **by** *auto*
have *4*: ⊢ *f ieqv* (*ae iand empty*) *ior*
big-ior (*map2* (λ *x y*. *x iand* (*next y*))
(*foldr* (*add-to-now-single*) *a* [*itrue*])
(*foldr* (*add-to-next-single*) *A* [*ifalse*]))
using *2 3* **by** *fastforce*
have *5*: *state-qpitl-list* (*foldr* (*add-to-now-single*) *a* [*itrue*])
by (*simp add*: *2 state-qpitl-foldr-add-to-now-single-inv-a*)
have *6*: *length* (*foldr* (*add-to-now-single*) *a* [*itrue*]) =
length (*foldr* (*add-to-next-single*) *A* [*ifalse*])
by (*simp add*: *2 length-add-to-now-next-gen*)
have *7*: *full-system* (*foldr* (*add-to-now-single*) *a* [*itrue*])
using *big-ior-foldr-add-to-now-single-inv-a full-system-def mutex-foldr-add-to-now-single-inv-a* **by** *blast*
show *?thesis* **unfolding** *GNF-prop-def GNF-cons-def*
using *2 4 5 6 7* **by** *blast*
qed

lemma *state-qpitl-list-GNF-det-state*:
state-qpitl-list (*foldr* (*add-to-now-single*) (*GNF-state f*) [*itrue*])
by (*simp add*: *state-qpitl-foldr-add-to-now-single-inv-a state-qpitl-list-GNF-state*)

lemma *full-system-GNF-det-state*:
full-system (*foldr* (*add-to-now-single*) (*GNF-state f*) [*itrue*])
using *big-ior-foldr-add-to-now-single-inv-a full-system-def*
mutex-foldr-add-to-now-single-inv-a **by** *auto*

lemma *ieqv-GNF-det*:
⊢ *f ieqv GNF-det f*
using *ieqv-GNF*[*of f*]
unfolding *GNF-def GNF-det-def GNF-cons-def*

using *det-big-ior-ieqv-nondet-big-ior-a length-GNF-state-eq-length-GNF-next* **by** *auto*

end

Chapter 6

CDT-PITL

theory *CDT-PITL-Deep*

imports
NELList-Extras
begin

This theory contains a deep embedding of PITL with the CDT operators of Yde Venema[17] without chop-star and chop-omega.

6.1 Syntax

6.1.1 Primitive formulae

datatype *cdt* =

false-d	(*ifalse*)
| *atom-d nat*	(($-) [*100*] *99*)
| *iimp-d cdt cdt*	((- *iimp* -) [*26*,*25*] *25*)
| *empty-d*	((*empty*))
| *c-d cdt cdt*	((- *cee* -) [*84*,*84*] *83*)
| *d-d cdt cdt*	((- *dee* -) [*84*,*84*] *83*)
| *t-d cdt cdt*	((- *tee* -) [*84*,*84*] *83*)

6.1.2 Derived Boolean Operators

definition *inot-d* ((*inot* -) [*90*] *90*)
where
inot f ≡ *f iimp ifalse*

definition *itrue-d* (*itrue*)
where
itrue ≡ *inot ifalse*

definition *ior-d* ((- *ior* -) [*31*,*30*] *30*)
where
f ior g ≡ (*inot f*) *iimp g*

definition *iand-d* ((- *iand* -) [*36*,*35*] *35*)
where
f iand g ≡ *inot* (*inot f ior inot g*)

definition *ieqv-d* ((- *ieqv* -) [*21*,*20*] *20*)
where
f ieqv g ≡ ((*f iimp g*) *iand* (*g iimp f*))

6.1.3 ifinite and inf

definition *finite-d* (*ifinite*)
where
ifinite ≡ *itrue cee empty*

definition *inf-d* (*inf*)
where
inf ≡ *inot ifinite*

6.1.4 weak chop

definition *chop-d* ((- ; -) [*84*,*84*] *83*)
where *f* ; *g* ≡ (*f cee g*) *ior* (*f iand inf*)

6.1.5 more, skip, next, wnext, prev and wprev

definition *more-d* (*more*)
where
more ≡ *inot empty*

definition *skip-d* (*skip*)
where
skip ≡ *more iand inot*(*more cee more*)

definition *next-d* ((*next* -))
where *next f* ≡ *skip cee f*

definition *wnext-d* ((*wnext* -))
where *wnext f* ≡ (*empty ior next f*)

definition *prev-d* ((*prev* -))
where *prev f* ≡ *f* ; *skip*

definition *wprev-d* ((*wprev* -))
where *wprev f* ≡ (*empty ior prev f*)

6.1.6 Box and Diamond Operators

definition *diamond-d* ((*diamond* -) [*88*] *87*)
where
diamond f ≡ *itrue cee f*

definition *box-d* ((*box* -) [*88*] *87*)
where
box f ≡ *inot* (*diamond* (*inot f*))

definition *df-d* ((*df* -) [*88*] *87*)
where
df f ≡ *f cee itrue*

definition *bf-d* ((*bf* -) [*88*] *87*)
where
bf f ≡ *inot* (*df* (*inot f*))

definition *ldiamond-d* ((*ldiamond* -) [*88*] *87*)
where *ldiamond f* ≡ *f dee itrue*

definition *lbox-d* ((*lbox* -) [*88*] *87*)
where
lbox f = *inot* (*ldiamond* (*inot f*))

definition *rdiamond-d* ((*rdiamond* -) [*88*] *87*)
where *rdiamond f* ≡ (*f tee ifinite*)

definition *rbox-d* ((*rbox* -) [*88*] *87*)
where
rbox f = *inot* (*rdiamond* (*inot f*))

6.1.7 Initial and Final Operators

definition *init-d* ((*init* -) [*88*] *87*)
where
init f ≡ ((*empty iand f*) *cee itrue*)

definition *fin-d* ((*fin* -) [*88*] *87*)
where
fin f ≡ *diamond* (*empty iand f*)

6.1.8 Horizontal

definition
hor q = ((*lbox* $*q*) *iand* (*box* $*q*) *iand* (*ifinite iimp rbox* (*fin* $*q*)))

6.1.9 Compass operators

definition *ediamond-d* ((*ediamond* -) [*88*] *87*)
where
ediamond f ≡ *more cee f*

definition *wdiamond-d* ((*wdiamond* -) [*88*] *87*)
where

wdiamond f ≡ more dee f

definition *sdiamond-d* ((*sdiamond* -) [*88*] *87*)
where
sdiamond f ≡f cee more

definition *ndiamond-d* ((*ndiamond* -) [*88*] *87*)
where
ndiamond f ≡ more tee f

definition *ewdiamond-d* ((*ewdiamond* -) [*88*] *87*)
where
ewdiamond f ≡ ediamond f ior f ior wdiamond f

definition *nsdiamond-d* ((*nsdiamond* -) [*88*] *87*)
where
nsdiamond f ≡ sdiamond f ior f ior ndiamond f

definition *ebox-d* ((*ebox* -) [*88*] *87*)
where
ebox f = inot (*ediamond* (*inot f*))

definition *wbox-d* ((*wbox* -) [*88*] *87*)
where
wbox f = inot (*wdiamond* (*inot f*))

definition *sbox-d* ((*sbox* -) [*88*] *87*)
where
sbox f = inot (*sdiamond* (*inot f*))

definition *nbox-d* ((*nbox* -) [*88*] *87*)
where
nbox f = inot (*ndiamond* (*inot f*))

definition *lex-d* ((*lex*))
where *lex ≡ wbox ifalse*

definition *rex-d* ((*rex*))
where *rex ≡ nbox ifalse*

definition *ewbox-d* ((*ewbox* -) [*88*] *87*)
where
ewbox f = inot (*ewdiamond* (*inot f*))

definition *nsbox-d* ((*nsbox* -) [*88*] *87*)
where
nsbox f = inot (*nsdiamond* (*inot f*))

definition *ahor-d* ((*ahor* -) [*88*] *87*)
where *ahor f ≡ ewbox f iand nbox* (*ewbox*(*inot f*)) *iand sbox*(*ewbox* (*inot f*))

definition *aver-d* ((*aver* -) [*88*] *87*)
where *aver f* ≡ *nsbox f iand ebox* (*nsbox*(*inot f*)) *iand wbox*(*nsbox* (*inot f*))

6.2 Semantics

6.2.1 Intervals

type-synonym *interval* = *nat set nellist*

fun *semantics-cdt* :: [*interval*, *nat*, *enat*, *cdt*] ⇒ *bool* ((- - - ⊨$_i$ -) [*80*,*10*] *10*)
where
(*g i j* ⊨$_i$ *ifalse*) = *False*
| (*g i j* ⊨$_i$ $*p*) = ((*enat i*)≤ *nlength g* ∧ *p* ∈ (*nnth g i*))
| (*g i j* ⊨$_i$ *F iimp G*) = ((*g i j* ⊨$_i$ *F*) ⟶ (*g i j* ⊨$_i$ *G*))
| (*g i j* ⊨$_i$ *empty*) = ((*enat i*) = *j*)
| (*g i j* ⊨$_i$ *F cee G*) = (∃ *k*. (*enat i*) ≤ (*enat k*) ∧ (*enat k*) ≤ *j* ∧ (*g i k* ⊨$_i$ *F*) ∧ (*g k j* ⊨$_i$ *G*))
| (*g i j* ⊨$_i$ *F tee G*) = (∃ *k l*. *j*< ∞ ∧ *l* = *the-enat j* ∧ *j* ≤ *k* ∧ *k* ≤ *nlength g* ∧ (*g l k* ⊨$_i$ *F*) ∧ (*g i k* ⊨$_i$ *G*))
| (*g i j* ⊨$_i$ *F dee G*) = (∃ *k*. *0* ≤ *k* ∧ *k* ≤ *i* ∧ (*g k i* ⊨$_i$ *F*) ∧ (*g k j* ⊨$_i$ *G*))

primrec *fvars* :: *cdt* ⇒ *nat set*
where
fvars ifalse = {}
| *fvars* ($*p*) = {*p*}
| *fvars* (*f iimp g*) = (*fvars f*) ∪ (*fvars g*)
| *fvars empty* = {}
| *fvars* (*f cee g*) = (*fvars f*) ∪ (*fvars g*)
| *fvars* (*f tee g*) = *fvars f* ∪ (*fvars g*)
| *fvars* (*f dee g*) = *fvars f* ∪ (*fvars g*)

definition *upd* :: *interval* ⇒ *nat* ⇒ (*bool nellist*) ⇒ *interval*
where
upd σ *n l* = *nmap* (λ (*x*,*y*). (*if y then insert n x else x* − {*n*})) (*nzip* σ *l*)

lemma *upd-NCons*:
upd (*NCons x xs*) *n* (*NCons y ys*) =
(*if y then NCons* (*insert n x*) (*upd xs n ys*) *else NCons* (*x* − {*n*}) (*upd xs n ys*))
by (*simp add*: *upd-def*)

lemma *upd-NNil*:
upd (*NNil a*) *n* (*NNil b*) =
(*if b then* (*NNil* (*insert n a*)) *else* (*NNil* (*a* − {*n*})))
by (*simp add*: *nfirst-eq-nnth-zero nnth-NNil upd-def*)

lemma *nnth-upd*:
assumes *nlength σ* = *nlength l*
enat k ≤ *nlength σ*
shows (*nnth* (*upd σ n l*) *k*) = (*if* (*nnth l k*) *then* (*insert n* (*nnth σ k*)) *else* (*nnth σ k*) − {*n*})
using *assms*
by (*simp add*: *upd-def nnth-nzip*)

lemma *nlength-upd*:
assumes *nlength σ* = *nlength l*
shows *nlength* ((*upd σ n l*)) = *nlength σ*
using *assms* **by** (*simp add*: *upd-def*)

lemma *upd-swap*:
assumes *nlength σ* = *nlength l1*
nlength σ = *nlength l2*
x1 ≠ *x2*
shows *upd* (*upd σ x1 l1*) *x2 l2* = (*upd* (*upd σ x2 l2*) *x1 l1*)
proof −
have *1*: *nlength* (*upd* (*upd σ x1 l1*) *x2 l2*) = *nlength* (*upd* (*upd σ x2 l2*) *x1 l1*)
using *assms nlength-upd* **by** *auto*
have *2*: ⋀ *i*. *i* ≤ *nlength* (*upd* (*upd σ x1 l1*) *x2 l2*) ⟹
(*nnth* (*upd* (*upd σ x1 l1*) *x2 l2*) *i*) =
(*if* (*nnth l2 i*) *then insert x2* (*nnth* (*upd σ x1 l1*) *i*) *else* (*nnth* (*upd σ x1 l1*) *i*) − {*x2*})
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *3*: ⋀ *i*. *i* ≤ *nlength* (*upd* (*upd σ x1 l1*) *x2 l2*) ⟹
(*nnth* (*upd* (*upd σ x2 l2*) *x1 l1*) *i*) =
(*if* (*nnth l1 i*) *then insert x1* (*nnth* (*upd σ x2 l2*) *i*) *else* (*nnth* (*upd σ x2 l2*) *i*) − {*x1*})
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *4*: ⋀ *i*. *i* ≤ *nlength* (*upd* (*upd σ x1 l1*) *x2 l2*) ⟹
(*if* (*nnth l2 i*) *then insert x2* (*nnth* (*upd σ x1 l1*) *i*) *else* (*nnth* (*upd σ x1 l1*) *i*) − {*x2*}) =
(*if* (*nnth l1 i*) *then insert x1* (*nnth* (*upd σ x2 l2*) *i*) *else* (*nnth* (*upd σ x2 l2*) *i*) − {*x1*})
using *assms nnth-upd nlength-upd* **by** *auto*
show *?thesis*
by (*simp add*: *1 2 3 4 nellist-eq-nnth-eq*)
qed

lemma *upd-absorb*:
assumes *nlength σ* = *nlength l1*
nlength σ = *nlength l2*
x1 = *x2*
shows *upd* (*upd σ x1 l1*) *x2 l2* = (*upd σ x2 l2*)
proof −
have *1*: *nlength* (*upd* (*upd σ x1 l1*) *x2 l2*) = *nlength* (*upd σ x2 l2*)

using *assms nlength-upd* **by** *auto*
have *2*: $\bigwedge i.\ i \leq nlength\ (upd\ (upd\ \sigma\ x1\ l1)\ x2\ l2) \Longrightarrow$
$(nnth\ (upd\ (upd\ \sigma\ x1\ l1)\ x2\ l2)\ i) =$
$(if\ (nnth\ l2\ i)\ then\ insert\ x2\ (nnth\ (upd\ \sigma\ x1\ l1)\ i)\ else\ (nnth\ (upd\ \sigma\ x1\ l1)\ i) - \{x2\})$
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *3*: $\bigwedge i.\ i \leq nlength\ (upd\ (upd\ \sigma\ x1\ l1)\ x2\ l2) \Longrightarrow$
$(nnth\ (upd\ \sigma\ x2\ l2)\ i) =$
$(if\ (nnth\ l2\ i)\ then\ insert\ x2\ (nnth\ \sigma\ i)\ else\ (nnth\ \sigma\ i) - \{x2\})$
using *assms* **by** (*simp add*: *nlength-upd nnth-upd*)
have *4*: $\bigwedge i.\ i \leq nlength\ (upd\ (upd\ \sigma\ x1\ l1)\ x2\ l2) \Longrightarrow$
$(if\ (nnth\ l2\ i)\ then\ insert\ x2\ (nnth\ (upd\ \sigma\ x1\ l1)\ i)\ else\ (nnth\ (upd\ \sigma\ x1\ l1)\ i) - \{x2\}) =$
$(if\ (nnth\ l2\ i)\ then\ insert\ x2\ (nnth\ \sigma\ i)\ else\ (nnth\ \sigma\ i) - \{x2\})$
using *assms nnth-upd nlength-upd* **by** *auto*
show *?thesis*
by (*simp add*: *1 2 3 4 nellist-eq-nnth-eq*)
qed

lemma *upd-nappend*:
assumes $nlength\ \sigma 1 = nlength\ l1$
$nlength\ \sigma 2 = nlength\ l2$
shows $upd\ (nappend\ \sigma 1\ \sigma 2)\ x\ (nappend\ l1\ l2) =$
$(nappend\ (upd\ \sigma 1\ x\ l1)\ (upd\ \sigma 2\ x\ l2))$
using *assms*
by (*simp add*: *nmap-nappend-distrib nzip-nappend upd-def*)

lemma *ntaken-upd*:
assumes $nlength\ \sigma = nlength\ l$
$j \leq nlength\ \sigma$
shows $ntaken\ j\ (upd\ \sigma\ x\ l) = (upd\ (ntaken\ j\ \sigma)\ x\ (ntaken\ j\ l))$
using *assms*
by (*simp add*: *ntaken-nzip upd-def*)

lemma *ndropn-upd*:
assumes $nlength\ \sigma = nlength\ l$
$j \leq nlength\ \sigma$
shows $ndropn\ j\ (upd\ \sigma\ x\ l) = (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l))$
using *assms ndropn-nzip upd-def* **by** (*simp add*: *ndropn-nmap*)

lemma *nsubn-upd*:
assumes $nlength\ \sigma = nlength\ l$
$(enat\ j) \leq enat\ k$
$(enat\ k) \leq nlength\ \sigma$
shows $nsubn\ (upd\ \sigma\ x\ l)\ j\ k\ = (upd\ (nsubn\ \ \sigma\ j\ k)\ x\ (nsubn\ \ l\ j\ k))$
proof −
have *1*: $nsubn\ (upd\ \sigma\ x\ l)\ j\ k = ntaken\ (k - j)\ (ndropn\ j\ (upd\ \sigma\ x\ l))$
using *assms* **unfolding** *nsubn-def1* **by** *simp*
have *2*: $(ndropn\ j\ (upd\ \sigma\ x\ l)) = (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l))$

```
  using assms by (meson dual-order.trans ndropn-upd)
 have 3:  ntaken (k − j) (upd (ndropn j σ) x (ndropn j l)) =
          (upd (ntaken (k−j) (ndropn j σ)) x (ntaken (k−j) (ndropn j l)))
   using assms  ntaken-upd[of (ndropn j σ) (ndropn j l) (k − j) x]
   by (metis ndropn-nlength nle-le nlength-upd ntaken-all)
 have 4: (upd (ntaken (k−j) (ndropn j σ)) x (ntaken (k−j) (ndropn j l))) =
         (upd (nsubn  σ j k) x (nsubn  l j k))
   unfolding nsubn-def1 by simp
 show ?thesis using 1 2 3 4 by presburger
qed

lemma nfirst-upd:
 assumes  nlength σ = nlength l
 shows nfirst (upd σ x l) = (if nfirst l then insert x (nfirst σ) else (nfirst σ) − {x} )
 by (metis assms nfirst-eq-nnth-zero nnth-upd zero-enat-def zero-le)

lemma nlast-upd:
 assumes  nlength σ = nlength l
         nfinite σ
 shows  nlast (upd σ x l) = (if nlast l then insert x (nlast σ) else (nlast σ) − {x} )
using assms
by (metis dual-order.order-iff-strict nfinite-conv-nlength-enat nlength-upd nnth-nlast nnth-upd the-enat.simps)

lemma nfinite-upd:
 assumes  nlength σ = nlength l
 shows   nfinite (upd σ x l) ⟷ nfinite σ ∧ nfinite l
using assms
by (simp add: nfinite-conv-nlength-enat nlength-upd)

lemma upd-nfuse:
 assumes nlength σ1 = nlength l1
         nlength σ2 = nlength l2
         nfirst σ2 = nlast σ1
          nfinite σ1
          nfirst l2 = nlast l1
 shows  upd (nfuse σ1 σ2) x (nfuse l1 l2) =
        (nfuse (upd σ1 x l1) (upd σ2 x l2))
using assms by (simp add: nfuse-nlength nmap-nfuse nzip-nfuse upd-def)

lemma nrev-upd:
  assumes nfinite g
          nlength g = nlength l
  shows nrev (upd g n l ) = (upd (nrev g) n (nrev l))
proof −
 have 1: nlength (nrev (upd g n l )) = nlength (upd (nrev g) n (nrev l))
  by (simp add: assms(2) nlength-upd)
 have 2: ⋀i. i ≤ nlength (nrev (upd g n l )) ⟶
          (nnth (nrev (upd g n l )) i) =
          (nnth (upd (nrev g) n (nrev l)) i)
 using assms by (auto simp add:  nlength-upd nnth-nrev nnth-upd nfinite-conv-nlength-enat)
```

show *?thesis*
using *1 2 nellist-eq-nnth-eq* **by** *blast*
qed

lemma *upd-exist*:
$\exists$ *r2 ur2. nlength r2 = nlength r1* $\wedge$ *nlength ur2 = nlength r1* $\wedge$ *r1 = upd r2 n ur2*
proof −
have *1*: $\exists$ *ur1. nlength ur1 = nlength r1* $\wedge$ ($\forall$ *i. i*$\leq$ *nlength r1* $\longrightarrow$ (*nnth ur1 i*) = (*n* $\in$ (*nnth r1 i*)))
using *nlength-nmap nnth-nmap* **by** *blast*
have *2*: $\exists$ *r3. nlength r3 = nlength r1* $\wedge$ ($\forall$ *i. i* $\leq$ *nlength r1* $\longrightarrow$ (*nnth r3 i*) = (*nnth r1 i*) −{*n*})
by (*metis nlength-nmap nnth-nmap Set.remove-eq*)
obtain *ur1* **where** *3*: *nlength ur1 = nlength r1* $\wedge$ ($\forall$ *i. i*$\leq$ *nlength r1* $\longrightarrow$ (*nnth ur1 i*) = (*n* $\in$ (*nnth r1 i*)))
using *1* **by** *blast*
obtain *r3* **where** *4*: *nlength r3 = nlength r1* $\wedge$ ($\forall$ *i. i* $\leq$ *nlength r1* $\longrightarrow$ (*nnth r3 i*) = (*nnth r1 i*) −{*n*})
using *2* **by** *blast*
have *5*: *r1 = upd r3 n ur1*
by (*metis 3 4 Diff-empty Diff-insert0 insert-Diff nellist-eq-nnth-eq nlength-upd nnth-upd*)
show *?thesis*
using *3 4 5* **by** *auto*
qed

lemma *not-fvar-upd*:
assumes *n* $\notin$ *fvars f*
nlength ug = nlength g
i $\leq$ *j*
j $\leq$ *nlength g*
shows (*g i j* $\models_i$ *f*) = ((*upd g n ug*) *i j* $\models_i$ *f*)
using *assms*
proof (*induction f arbitrary*: *g i j ug*)
case *false-d*
then show *?case* **by** *auto*
next
case (*atom-d x*)
then show *?case*
by (*simp add*: *nlength-upd nnth-upd*)
next
case (*iimp-d f1 f2*)
then show *?case* **by** *simp*
next
case *empty-d*
then show *?case* **using** *nlength-upd* **by** *auto*
next
case (*c-d f1 f2*)
then show *?case*
proof −
have *c1*: (*g i j* $\models_i$ *f1 cee f2*) =
(∃ *k*$\geq$*i. enat k* $\leq$ *j* $\wedge$ (*g i* (*enat k*) $\models_i$ *f1*) $\wedge$ (*g k j* $\models_i$ *f2*))

```
   by simp
   have c2: ((upd g n ug) i j ⊨ᵢ  f1 cee f2 ) =
          (∃ k≥i. enat k ≤ j ∧ ((upd g n ug) i (enat k) ⊨ᵢ f1) ∧ ((upd g n ug) k j ⊨ᵢ f2))
    by simp
   have c3: (∃ k≥i. enat k ≤ j ∧ (g i (enat k) ⊨ᵢ f1) ∧ (g k j ⊨ᵢ f2)) ⟹
          (∃ k≥i. enat k ≤ j ∧ ((upd g n ug) i (enat k) ⊨ᵢ f1) ∧ ((upd g n ug) k j ⊨ᵢ f2))
   using c-d by auto
  have c5: (∃ k≥i. enat k ≤ j ∧ ((upd g n ug) i (enat k) ⊨ᵢ f1) ∧ ((upd g n ug) k j ⊨ᵢ f2))
          ⟹ (∃ k≥i. enat k ≤ j ∧ (g i (enat k) ⊨ᵢ f1) ∧ (g k j ⊨ᵢ f2))
   using c-d by auto
   show ?thesis   using c1 c2 c3 c5 by argo
  qed
next
case (d-d f1 f2)
then show ?case
   proof −
    have d1: (g i j ⊨ᵢ  f1 dee f2 ) =
            (∃ k≤i. (g k (enat i) ⊨ᵢ f1) ∧ (g k j ⊨ᵢ f2))
     by simp
    have d2: ((upd g n ug) i j ⊨ᵢ  f1 dee f2 ) =
            (∃ k≤i. ((upd g n ug) k (enat i) ⊨ᵢ f1) ∧ ((upd g n ug) k j ⊨ᵢ f2))
     by simp
    have d3: (∃ k≤i. (g k (enat i) ⊨ᵢ f1) ∧ (g k j ⊨ᵢ f2)) ⟹
            (∃ k≤i. ((upd g n ug) k (enat i) ⊨ᵢ f1) ∧ ((upd g n ug) k j ⊨ᵢ f2))
    using d-d
     by (metis Un-iff dual-order.trans fvars.simps(7) of-nat-eq-enat of-nat-mono)
    have d4: (∃ k≤i. ((upd g n ug) k (enat i) ⊨ᵢ f1) ∧ ((upd g n ug) k j ⊨ᵢ f2)) ⟹
            (∃ k≤i. (g k (enat i) ⊨ᵢ f1) ∧ (g k j ⊨ᵢ f2))
    using d-d
    by (metis Un-iff  dual-order.trans enat-ord-simps(1) fvars.simps(7))
    show ?thesis
    using d1 d2 d3 d4 by argo
   qed
next
case (t-d f1 f2)
then show ?case
  proof −
   have t1: (g i j ⊨ᵢ  f1 tee f2 ) =
            ((∃ i. j = enat i) ∧ (∃ k≥j. k ≤ nlength g ∧ (g (the-enat j) k ⊨ᵢ f1) ∧ (g i k ⊨ᵢ f2)))
   by simp
  have t2: ((upd g n ug) i j ⊨ᵢ  f1 tee f2 ) =
     ((∃ i. j = enat i) ∧ (∃ k≥j. k ≤ nlength (upd g n ug) ∧ ((upd g n ug) (the-enat j) k ⊨ᵢ f1) ∧ ((upd g n
ug) i k ⊨ᵢ f2)))
   by simp
  have t3: ((∃ i. j = enat i) ∧ (∃ k≥j. k ≤ nlength g ∧ (g (the-enat j) k ⊨ᵢ f1) ∧ (g i k ⊨ᵢ f2))) ⟹
          ((∃ i. j = enat i) ∧ (∃ k≥j. k ≤ nlength (upd g n ug) ∧ ((upd g n ug) (the-enat j) k ⊨ᵢ f1) ∧ ((upd
g n ug) i k ⊨ᵢ f2)))
   using t-d by simp
    (metis (no-types, lifting) nlength-upd order-trans the-enat.simps)
  have t4: ((∃ i. j = enat i) ∧ (∃ k≥j. k ≤ nlength (upd g n ug) ∧ ((upd g n ug) (the-enat j) k ⊨ᵢ f1) ∧
```

```
((upd g n ug) i k ⊨ᵢ f2)))
          ⟹
          ((∃ i. j = enat i) ∧ (∃ k≥j. k ≤ nlength g ∧ (g (the-enat j) k ⊨ᵢ f1) ∧ (g i k ⊨ᵢ f2)))
  using  t-d by simp
   (metis dual-order.trans nlength-upd the-enat.simps)
  show ?thesis
  using t1 t2 t3 t4 by argo
 qed
qed

lemma inot-defs :
 shows  (g i j ⊨ᵢ inot f ) = Not (g i j ⊨ᵢ f)
by (simp add: inot-d-def)

lemma ior-defs :
   (g i j ⊨ᵢ f1 ior f2) = ( (g i j ⊨ᵢ f1) ∨ (g i j ⊨ᵢ f2))
by (metis inot-defs ior-d-def semantics-cdt.simps(3))

lemma iand-defs :
    (g i j ⊨ᵢ f1 iand f2) = ( (g i j ⊨ᵢ f1) ∧ (g i j ⊨ᵢ f2))
  using iand-d-def inot-defs ior-defs by presburger

lemma ieqv-defs :
   (g i j ⊨ᵢ f1 ieqv f2) = ( (g i j ⊨ᵢ f1) = (g i j ⊨ᵢ f2))
  using iand-defs ieqv-d-def by auto

lemma itrue-defs :
   (g i j ⊨ᵢ itrue)
by (simp add: inot-d-def itrue-d-def)

lemma empty-defs :
   (g i j ⊨ᵢ empty) = (i = j)
by simp

lemma more-defs:
  (g i j ⊨ᵢ more) ⟷ (i ≠ j)
 unfolding more-d-def
 using empty-defs i0-less inot-defs by presburger

lemma skip-defs:
assumes i ≤j
        j ≤ nlength g
shows  (g i j ⊨ᵢ skip) ⟷ j = enat (Suc i)
proof −
 have 1: (g i j ⊨ᵢ skip) ⟷ (g i j ⊨ᵢ more iand inot ( more cee more ))
  unfolding skip-d-def by blast
 have 2: (g i j ⊨ᵢ more iand inot ( more cee more )) ⟷
         (g i j ⊨ᵢ more) ∧ (g i j ⊨ᵢ inot ( more cee more ))
   using  iand-defs by presburger
 have 3:  (g i j ⊨ᵢ inot ( more cee more )) ⟷ Not (g i j ⊨ᵢ  ( more cee more ))
```

using *inot-defs* **by** *presburger*
have *4*: $(g\ i\ j \models_i\ (\ more\ cee\ more\ )) \longleftrightarrow$
$(\exists\, k{\geq}i.\ enat\ k \leq j \wedge (g\ i\ (enat\ k) \models_i more) \wedge (g\ k\ j \models_i more))$
by *simp*
have *5*: $(\exists\, k{\geq}i.\ enat\ k \leq j \wedge (g\ i\ (enat\ k) \models_i more) \wedge (g\ k\ j \models_i more)) \longleftrightarrow$
$(\exists\, k{\geq}i.\ enat\ k \leq j \wedge i \neq (enat\ k) \wedge k \neq j\ )$
using *more-defs* **by** *presburger*
have *7*: $(\exists\, k{\geq}i.\ enat\ k \leq j \wedge i \neq (enat\ k) \wedge k \neq j\ ) \Longrightarrow$
$j > enat\ (Suc\ i)$
by (*metis Suc-ile-eq assms*(*1*) *enat-ord-simps*(*1*) *nle-le order-neq-le-trans*)
have *8*: $j > enat\ (Suc\ i) \Longrightarrow (\exists\, k{\geq}i.\ enat\ k \leq j \wedge i \neq (enat\ k) \wedge k \neq j\ )$
by (*metis enat.inject le-add2 linorder-neq-iff n-not-Suc-n order-less-imp-le plus-1-eq-Suc*)
have *9*: $(\exists\, k{\geq}i.\ enat\ k \leq j \wedge i \neq (enat\ k) \wedge k \neq j\ ) \longleftrightarrow$
$j > enat\ (Suc\ i)$
using *7 8* **by** *blast*
have *10*: $(g\ i\ j \models_i\ (\ more\ cee\ more\ )) \longleftrightarrow (j > enat\ (Suc\ i))$
using *4 5 7 8* **by** *argo*
have *11*: $(g\ i\ j \models_i more) \wedge (g\ i\ j \models_i inot\ (\ more\ cee\ more\ )) \longleftrightarrow$
$i \neq j \wedge Not(j > enat\ (Suc\ i))$
using *10 3 more-defs* **by** *presburger*
have *12*: $i \neq j \wedge Not(j > enat\ (Suc\ i)) \longleftrightarrow\ j = enat\ (Suc\ i)$
by (*metis Suc-ile-eq assms*(*1*) *dual-order.irrefl order-neq-le-trans*)
show *?thesis*
using *1 11 12 2* **by** *presburger*
qed

lemma *skip-cee-itrue-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i (skip\ cee\ itrue)) = (i < j)$
proof −
have *1*: $(g\ i\ j \models_i (skip\ cee\ itrue)) \Longrightarrow (i < j)$
using *assms nless-le skip-defs* **by** *fastforce*
have *2*: $(i < j) \Longrightarrow (g\ i\ j \models_i (skip\ cee\ itrue))$
using *assms*
by (*metis eSuc-enat ileI1 ile-eSuc itrue-defs order.trans semantics-cdt.simps*(*5*) *skip-defs*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *itrue-cee-skip-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i (itrue\ cee\ skip)) = (i < j \wedge (\exists\ k.\ j = (enat\ k)))$
proof −
have *1*: $(g\ i\ j \models_i (itrue\ cee\ skip)) \Longrightarrow (i < j \wedge (\exists\ k.\ j = (enat\ k)))$
using *assms* **by** *simp*
(*metis enat-ord-simps*(*1*) *iand-defs more-defs nle-le order-neq-le-trans skip-d-def skip-defs*)
have *2*: $(i < j \wedge (\exists\ k.\ j = (enat\ k))) \Longrightarrow (g\ i\ j \models_i (itrue\ cee\ skip))$
proof −
assume *a0*: $(i < j \wedge (\exists\ k.\ j = (enat\ k)))$

```
    show (g i j ⊨i (itrue cee skip))
    proof −
     have 21: (g i j ⊨i (itrue cee skip)) = (∃ k≥i. enat k ≤ j  ∧ j = (enat (Suc k)))
       using assms
       by (meson enat-ord-simps(1) itrue-defs semantics-cdt.simps(5) skip-defs)
     have 22: (∃ k≥i. enat k ≤ j  ∧ j = (enat (Suc k)))
      using assms a0
      using Nat.lessE by fastforce
     show ?thesis
     using 21 22 by blast
   qed
  qed
 show ?thesis using 1 2 by blast
qed

lemma ifinite-defs:
 assumes i ≤ j
         j ≤ nlength g
 shows   (g i j ⊨i ifinite) ⟷ (∃ k. j = (enat k))
using assms  unfolding finite-d-def itrue-d-def inot-d-def
by auto


lemma inf-defs:
assumes  i ≤ j
         j ≤ nlength g
shows (g i j ⊨i inf) ⟷ j = ∞
using assms
unfolding inf-d-def by (simp add: ifinite-defs inot-defs)

lemma chop-defs:
 assumes  i ≤ j
         j ≤ nlength g
 shows (g i j ⊨i f1 ; f2 )  =
         (∃  k. (enat i) ≤ (enat k) ∧ (enat k) ≤ j ∧ ( g i k ⊨i f1) ∧ ( g k j ⊨i f2) ) ∨
         ( j = ∞ ∧ ( g i j ⊨i f1) )
using assms apply (simp add: iand-defs ior-defs chop-d-def inf-defs)
by auto

lemma diamond-defs:
assumes  i ≤ j
         j ≤ nlength g
 shows (g i j ⊨i diamond f) ⟷
        (∃  k. i ≤ k ∧ k ≤ j ∧ ( g k j  ⊨i f))
using assms
unfolding diamond-d-def
by (simp add: inot-defs itrue-defs)

lemma box-defs:
assumes  i ≤ j
```

$j \leq nlength\ g$
shows $(g\ i\ j \models_i box\ f) \longleftrightarrow$
$(\forall\ k.\ i \leq k \wedge k \leq j \longrightarrow (\ g\ k\ j\ \models_i f))$
using *assms* **unfolding** *box-d-def*
by (*simp add*: *inot-defs diamond-defs*)
blast

lemma *fin-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i (fin\ \$p)) \longleftrightarrow ((\exists\ k.\ j = (enat\ k)) \wedge p \in (nnth\ g\ (the\text{-}enat\ j)))$
using *assms*
unfolding *fin-d-def*
by (*auto simp add*: *diamond-defs iand-defs*)

lemma *df-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i df\ f) \longleftrightarrow$
$(\exists\ k.\ i \leq k \wedge k \leq j\ \wedge (g\ i\ (enat\ k) \models_i f))$
using *assms*
unfolding *df-d-def*
by (*simp add*: *inot-defs itrue-defs*)

lemma *atom-diamond-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i diamond\ \$q) \longleftrightarrow (\exists\ k.\ i \leq k \wedge k \leq j \wedge q \in (nnth\ g\ k))$
using *assms diamond-defs*
by *auto*

lemma *atom-df-fin-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i df\ (fin\ \$q)) \longleftrightarrow (\exists\ k.\ i \leq k \wedge k \leq j\ \wedge q \in (nnth\ g\ k))$
using *assms df-defs fin-defs*
by *auto*

lemma *atom-box-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i box\ \$q) \longleftrightarrow (\forall\ k.\ i \leq k \wedge k \leq j\ \longrightarrow q \in (nnth\ g\ k))$
using *assms box-defs*
by *auto*

lemma *atom-fin-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i\ fin\ \$q) \longleftrightarrow ((\exists k.\ j = (enat\ k)\ \wedge q \in (nnth\ g\ (the\text{-}enat\ j))))$
using *assms*

using *fin-defs nnth-nlast* **by** *blast*

lemma *atom-lbox-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i (lbox\ \$q)\ ) \longleftrightarrow$
$(\forall\ k.\ \ k \leq i\ \longrightarrow q \in (nnth\ g\ k))$
using *assms* **by** (*auto simp add*: *inot-defs lbox-d-def ldiamond-d-def itrue-defs*)
(*meson enat-ord-simps*(*1*) *order-trans*)

lemma *atom-rbox-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ifinite\ iimp\ rbox\ (fin\ \$q)\ ) \longleftrightarrow$
$(\ (\exists\ k\ .\ j = (enat\ k)) \longrightarrow\ (\forall\ k.\ j \leq k \wedge k \leq nlength\ g \longrightarrow q \in (nnth\ g\ k)))$
using *assms*
by (*auto simp add*: *rbox-d-def rdiamond-d-def inot-defs itrue-defs iand-defs ifinite-defs*
fin-d-def diamond-d-def)

lemma *hor-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i (hor\ q)) \longleftrightarrow$
$(\forall\ k.\ k \leq nlength\ g \longrightarrow q \in (nnth\ g\ k))$
using *assms*
using *atom-box-defs hor-def iand-defs atom-lbox-defs atom-rbox-defs*
by *simp*
(*meson dual-order.trans enat-ile enat-ord-simps*(*1*) *nle-le*)

lemma *ediamond-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ediamond\ f) = (\exists k.\ i < k \wedge enat\ k \leq j \wedge (g\ k\ j \models_i f))$
using *assms*
apply (*simp add*: *ediamond-d-def more-d-def inot-defs*)
using *dual-order.strict-iff-order* **by** *blast*

lemma *wdiamond-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i wdiamond\ f) = (\exists k.\ k < i \wedge (g\ k\ j \models_i f))$
using *assms*
apply (*simp add*: *wdiamond-d-def more-d-def inot-defs*)
using *order.order-iff-strict* **by** *blast*

lemma *sdiamond-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i sdiamond\ f) = (\exists k.\ i \leq k \wedge\ \ enat\ k < j \wedge (g\ i\ (enat\ k) \models_i f)\ )$
using *assms*

apply (*simp add*: *sdiamond-d-def more-d-def inot-defs*)
by (*metis linorder-neq-iff order-le-imp-less-or-eq order-less-imp-le*)

lemma *ndiamond-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ndiamond\ f) =$
$(j < \infty \wedge (\exists k.\ j < k \wedge k \leq nlength\ g \wedge (g\ i\ k \models_i f)))$
using *assms*
apply (*simp add*: *ndiamond-d-def more-d-def inot-defs*)
by (*metis linorder-neq-iff order-le-imp-less-or-eq order-less-imp-le the-enat.simps*)

lemma *ewdiamond-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ewdiamond\ f) =$
$(\exists k.\ enat\ k \leq j \wedge (g\ k\ j \models_i f))$
using *assms*
apply (*simp add*: *ewdiamond-d-def ediamond-defs wdiamond-defs ior-defs*)
by (*metis enat-ord-simps*(*2*) *linorder-less-linear order-le-less order-less-le-subst2*)

lemma *nsdiamond-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i nsdiamond\ f) =$
$(\ (\exists k.\ i \leq k \wedge k \leq nlength\ g \wedge (g\ i\ k \models_i f)\ )\ )$
using *assms*
apply (*auto simp add*: *ior-defs nsdiamond-d-def ndiamond-defs sdiamond-defs*)
apply (*meson dual-order.trans enat-ord-simps*(*1*) *order-less-imp-le*)
apply (*metis assms*(*1*) *dual-order.trans order-less-imp-le*)
apply (*metis enat-ord-simps*(*1*) *enat-ord-simps*(*4*) *ifinite-defs inf-d-def inf-defs inot-defs*)
by (*metis enat-ord-simps*(*1*) *ifinite-defs inf-d-def inf-defs inot-defs le-iff-add*
not-less-iff-gr-or-eq order-neq-le-trans plus-eq-infty-iff-enat)

lemma *ebox-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ebox\ f) = (\forall k.\ i < k \wedge enat\ k \leq j \longrightarrow (g\ k\ j \models_i f))$
using *assms ediamond-defs ebox-d-def inot-defs* **by** *auto*

lemma *ebox-ifalse-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ebox\ ifalse) = (i = j)$
using *assms ebox-defs*[*of i j g ifalse*]
by (*metis Suc-ile-eq antisym-conv2 enat-ord-simps*(*1*) *leD lessI semantics-cdt.simps*(*1*))

lemma *wbox-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i wbox\ f) = (\forall k.\ k < i \longrightarrow (g\ k\ j \models_i f))$
using *assms wdiamond-defs wbox-d-def inot-defs* **by** *auto*

lemma *lex-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i lex) = (i=0)$
using *assms* **using** *lex-d-def wbox-defs*
by (*metis antisym-conv1 le-add1 le-add-same-cancel1 less-nat-zero-code semantics-cdt.simps*(*1*))

lemma *sbox-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i sbox\ f) = (\forall k.\ i \leq k \wedge\ \ enat\ k < j \longrightarrow (g\ i\ (enat\ k) \models_i f)\ )$
using *assms sdiamond-defs sbox-d-def inot-defs* **by** *auto*

lemma *sbox-ifalse-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i sbox\ ifalse) = (i = j)$
using *assms sbox-defs*[*of i j g ifalse*]
by *fastforce*

lemma *nbox-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i nbox\ f) =$
$(j < \infty \longrightarrow (\forall k.\ j< k\ \wedge k \leq nlength\ g\ \longrightarrow (g\ i\ k \models_i f)))$
using *assms ndiamond-defs nbox-d-def inot-defs* **by** *auto*

lemma *rex-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i rex) = (j = nlength\ g)$
using *assms*
by (*metis enat-ord-simps*(*4*) *rex-d-def linorder-not-le nbox-defs order.order-iff-strict semantics-cdt.simps*(*1*))

lemma *ewbox-defs*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ewbox\ f) =$
$(\forall k.\ \ enat\ k \leq j \longrightarrow (g\ k\ j \models_i f))$
using *assms ewdiamond-defs ewbox-d-def inot-defs* **by** *auto*

lemma *ebox-ifalse-iand-lex-defs*:
assumes $i \leq j$
$j \leq nlength\ g$

shows $(g\ i\ j \models_i \mathit{ebox\ ifalse\ iand\ lex}) = (i = 0 \wedge i = j)$
using *assms ebox-ifalse-defs iand-defs lex-d-def lex-defs zero-enat-def* **by** *auto*

lemma *nsbox-defs*:
assumes $i \leq j$
$j \leq \mathit{nlength}\ g$
shows $(g\ i\ j \models_i \mathit{nsbox}\ f) =$
$(\ (\forall k.\ i \leq k \wedge k \leq \mathit{nlength}\ g \longrightarrow (g\ i\ k \models_i f)\ )\ )$
using *assms nsdiamond-defs nsbox-d-def inot-defs* **by** *auto*

lemma *rex-iand-sbox-ifalse-defs*:
assumes $i \leq j$
$j \leq \mathit{nlength}\ g$
shows $(g\ i\ j \models_i \mathit{rex\ iand\ sbox\ ifalse}) = (i = j \wedge j = \mathit{nlength}\ g)$
using *assms iand-defs rex-d-def rex-defs sbox-ifalse-defs* **by** *auto*

lemma *nbox-ewbox-inot-defs*:
assumes $i \leq j$
$j \leq \mathit{nlength}\ g$
shows $(g\ i\ j \models_i \mathit{nbox}\ (\mathit{ewbox\ inot}\ f)) =$
$(j < \infty \longrightarrow$
$(\forall k.\ j < k \wedge k \leq \mathit{nlength}\ g \longrightarrow$
$(\forall k1.\ \ \mathit{enat}\ k1 \leq k \longrightarrow \neg(g\ k1\ k \models_i f))$
$)\ )$
using *assms ewbox-defs inot-defs nbox-defs*[*of i j g* (*ewbox inot f*)]
apply *simp*
by (*meson order.order-iff-strict order-trans*)

lemma *sbox-ewbox-inot-defs*:
assumes $i \leq j$
$j \leq \mathit{nlength}\ g$
shows $(g\ i\ j \models_i \mathit{sbox}\ (\mathit{ewbox\ inot}\ f)) =$
$(\forall k.\ i \leq k \wedge \mathit{enat}\ k < j \longrightarrow$
$(\forall k1.\ \ \mathit{enat}\ k1 \leq (\mathit{enat}\ k) \longrightarrow \neg\ (g\ k1\ (\mathit{enat}\ k) \models_i\ \ f))$
$)$
using *assms sbox-defs*[*of i j g* (*ewbox inot f*)]
using *ewbox-defs inot-defs* **by** *auto*

lemma *ahor-defs*:
assumes $i \leq j$
$j \leq \mathit{nlength}\ g$
shows $(g\ i\ j \models_i \mathit{ahor}\ f) =$
$(\ (\forall k.\ \ \mathit{enat}\ k \leq j \longrightarrow (g\ k\ j \models_i f)) \wedge$
$(j < \infty \longrightarrow$
$(\forall k.\ j < k \wedge k \leq \mathit{nlength}\ g \longrightarrow$
$(\forall k1.\ \ \mathit{enat}\ k1 \leq k \longrightarrow \neg(g\ k1\ k \models_i f))$
$)\ )\ \wedge$
$(\forall k.\ i \leq k \wedge \mathit{enat}\ k < j \longrightarrow$
$(\forall k1.\ \ \mathit{enat}\ k1 \leq (\mathit{enat}\ k) \longrightarrow \neg(g\ k1\ (\mathit{enat}\ k) \models_i f))$

)
)
using *assms ahor-d-def ewbox-defs nbox-ewbox-inot-defs sbox-ewbox-inot-defs*
using *iand-defs* **by** *presburger*

lemma *ahor-defs-alt*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i ahor\ f) =$
$(\forall\ k\ k1\ .\ i \leq k \wedge\ enat\ k1 \leq k \wedge k \leq nlength\ g \longrightarrow (k = j \longleftrightarrow (g\ k1\ k \models_i f)))$
using *assms ahor-defs* **apply** *auto*
apply (*metis enat-ord-simps(1) enat-ord-simps(4) less-infinityE*)
apply (*metis enat-ord-simps(1) enat-ord-simps(4) leI less-infinityE nle-le*)
apply (*metis assms(1) dual-order.strict-trans2 less-le-not-le*)
by (*metis dual-order.trans enat-ord-simps(1) linorder-neq-iff order-less-imp-le*)

6.2.2 Validity

definition *valid* :: *cdt* $\Rightarrow$ *bool* $((\vdash_i -)\ 5)$
where $(\vdash_i f) = (\forall\ g\ i\ j.\ (enat\ i) \leq j \wedge j \leq nlength\ g \longrightarrow (g\ i\ j \models_i f))$

lemma *cdt-valid* [*simp*] :
$(\vdash_i f) = (\forall\ g\ i\ j.\ (enat\ i) \leq j \wedge j \leq nlength\ g \longrightarrow (g\ i\ j \models_i f))$
by (*simp add*: *valid-def*)

lemma *cdt-ieq*:
$(\vdash_i f1\ ieqv\ f2) = (\forall\ g\ i\ j.\ i \leq j \wedge j \leq nlength\ g \longrightarrow ((g\ i\ j \models_i f1) = (g\ i\ j \models_i f2)))$
using *ieqv-defs valid-def* **by** *presburger*

lemma *more-iand-help*:
assumes $i \leq j$
$j \leq nlength\ g$
shows $g\ i\ j \models_i (more\ iand\ ifinite)\ iimp\ (skip\ cee\ itrue)\ iand\ (itrue\ cee\ skip)$
using *assms skip-cee-itrue-defs itrue-cee-skip-defs iand-defs more-defs ifinite-defs*
apply *simp*
by *force*

lemma *Venema-discrete*:
$\vdash_i (more\ iand\ ifinite)\ iimp\ (skip\ cee\ itrue)\ iand\ (itrue\ cee\ skip)$
using *more-iand-help valid-def* **by** *presburger*

lemma *Venema-B1a*:
$\vdash_i (f1\ ior\ f2)\ cee\ g\ ieqv\ (f1\ cee\ g\ ior\ f2\ cee\ g)$
unfolding *valid-def*
by (*auto simp add*: *ieqv-defs ior-defs*)

lemma *Venema-B1b*:

$\vdash_i$ *f cee* (*g1 ior g2*) *ieqv* (*f cee g1 ior f cee g2*)
unfolding *valid-def*
by (*auto simp add*: *ieqv-defs ior-defs*)

lemma *Venema-B2a*:
$\vdash_i$ (*f1 ior f2*) *tee g ieqv* (*f1 tee g ior f2 tee g*)
unfolding *valid-def*
by (*auto simp add*: *ieqv-defs ior-defs*)

lemma *Venema-B2b*:
$\vdash_i$ *f tee* (*g1 ior g2*) *ieqv* (*f tee g1 ior f tee g2*)
unfolding *valid-def*
by (*auto simp add*: *ieqv-defs ior-defs*)

lemma *Venema-B3a*:
$\vdash_i$ (*f1 ior f2*) *dee g ieqv* (*f1 dee g ior f2 dee g*)
unfolding *valid-def*
by (*auto simp add*: *ieqv-defs ior-defs*)

lemma *Venema-B3b*:
$\vdash_i$ *f dee* (*g1 ior g2*) *ieqv* (*f dee g1 ior f dee g2*)
unfolding *valid-def*
by (*auto simp add*: *ieqv-defs ior-defs*)

lemma *Venema-C1*:
$\vdash_i$ (*inot*(*f tee g*)) *cee f iand ifinite iimp inot g*
by (*auto simp add*: *inot-defs iand-defs ifinite-defs*)

lemma *Venema-C2*:
$\vdash_i$ (*inot*(*f tee g*)) *dee g iimp inot* (*f*)
by (*auto simp add*: *inot-defs iand-defs inf-d-def ifinite-defs*)

lemma *Venema-C3*:
$\vdash_i$ *f tee* (*inot* (*g cee f*)) *iimp inot g*
by (*auto simp add*: *inot-defs*)

lemma *Venema-D*:
$\vdash_i$ *more cee itrue ieqv more*
by (*auto simp add*: *ieqv-defs more-defs inot-defs itrue-defs skip-defs*)
(*metis Suc-ile-eq Suc-n-not-le-n linorder-linear order.strict-iff-order*)

lemma *Venema-E1*:
$\vdash_i$ *empty cee f ieqv f*
by (*auto simp add*: *ieqv-defs*)

lemma *Venema-E2*:
$\vdash_i$ *empty tee f ieqv* (*f iand ifinite*)
by (*auto simp add*: *ifinite-defs ieqv-defs iand-defs*)

lemma *Venema-E3*:

$\vdash_i$ *f tee empty ieqv* (*f iand empty*)
by (*auto simp add*: *ieqv-defs iand-defs ifinite-defs*)

lemma *Venema-E4*:
$\vdash_i$ *empty dee f ieqv f*
by (*auto simp add*: *ieqv-defs*)

lemma *Venema-E5*:
$\vdash_i$ *f cee empty ieqv* (*f iand ifinite*)
by (*auto simp add*: *ifinite-defs ieqv-defs iand-defs*)

lemma *Venema-F*:
$\vdash_i$ ((*empty iand f*) *cee itrue iand* ((*empty iand g*) *cee itrue*) *cee itrue*) *iimp* (*empty iand g*) *cee itrue*
by (*auto simp add*: *iand-defs itrue-defs*)

lemma *cee-assoc-1*:
assumes (*g i j* $\models_i$ *f1 cee* (*f2 cee f3*))
shows (*g i j* $\models_i$ (*f1 cee f2*) *cee f3*)
using *assms dual-order.trans* **by** *auto*

lemma *cee-assoc-2*:
assumes (*g i j* $\models_i$ (*f1 cee f2*) *cee f3*)
shows (*g i j* $\models_i$ *f1 cee* (*f2 cee f3*))
using *assms*
by (*meson order.trans semantics-cdt.simps*(*5*))

lemma *Venema-G1*:
$\vdash_i$ *f cee* (*g cee h*) *ieqv* (*f cee g*) *cee h*
unfolding *valid-def ieqv-defs*
using *cee-assoc-1 cee-assoc-2*
by *blast*

lemma *Venema-G2a*:
$\vdash_i$ *f tee* (*g cee h*) *iimp* (*g cee* (*f tee h*) *ior* (*h tee f*) *tee g*)
by (*simp add*: *ior-defs*)
(*metis nle-le order-trans the-enat.simps*)

lemma *Venema-G2b*:
$\vdash_i$ (*g cee* (*f tee h*) *ior* (*h tee f*) *tee g*) *iimp f tee* (*g cee h*)
by (*simp add*: *ior-defs*)
(*metis enat-ord-simps*(*1*) *order-trans the-enat.simps*)

lemma *Venema-G3*:
$\vdash_i$ *f cee* (*g tee h*) *iimp g tee* (*f cee h*)
by (*auto*)
(*meson enat-ord-simps*(*1*) *order.trans*)

lemma *Venema-G4*:
$\vdash_i$ ((*f iand ifinite*) *tee g*) *cee h iimp* ((*h dee f*) *tee g ior g cee* (*f dee h*))
unfolding *valid-def*

apply (*auto simp add*: *ior-defs iand-defs ifinite-defs*)
using *ifinite-defs* **apply** *fastforce*
by (*metis dual-order.trans enat-ile enat-ord-simps*(*1*) *nle-le the-enat.simps*)

lemma *Venema-MP*:
assumes $\vdash_i f$
$\vdash_i f$ *iimp* g
shows $\vdash_i g$
using *assms* **unfolding** *valid-def* **by** *simp*

lemma *Venema-Gen-1a*:
assumes $\vdash_i f$
shows $\vdash_i$ *inot*((*inot f*) *cee g*)
using *assms* **unfolding** *valid-def*
apply (*simp add*: *inot-defs*)
by *force*

lemma *Venema-Gen-1b*:
assumes $\vdash_i g$
shows $\vdash_i$ *inot*(*g cee* (*inot g*))
using *assms* **unfolding** *valid-def*
by (*simp add*: *inot-defs*)

lemma *Venema-Gen-2a*:
assumes $\vdash_i f$
shows $\vdash_i$ *inot*((*inot f*) *tee g*)
using *assms* **unfolding** *valid-def*
apply (*simp add*: *inot-defs*)
by *force*

lemma *Venema-Gen-2b*:
assumes $\vdash_i g$
shows $\vdash_i$ *inot*(*f tee* (*inot g*))
using *assms* **unfolding** *valid-def*
apply (*simp add*: *inot-defs*)
by (*meson dual-order.trans*)

lemma *Venema-Gen-3a*:
assumes $\vdash_i f$
shows $\vdash_i$ *inot*((*inot f*) *dee g*)
using *assms* **unfolding** *valid-def*
apply (*simp add*: *inot-defs*)
by *force*

lemma *Venema-Gen-3b*:
assumes $\vdash_i g$
shows $\vdash_i$ *inot*(*f dee* (*inot g*))
using *assms* **unfolding** *valid-def*
apply (*simp add*: *inot-defs*)

by (*meson enat-ord-simps(1) order-trans*)

lemma *hor-rule*:
assumes $q \notin fvars\ f$
$\forall\ g\ i\ j.$
$(\ i \leq j \wedge j \leq nlength\ g \wedge$
$(\forall\ k.\ k \leq nlength\ g \longrightarrow q \in (nnth\ g\ k))\ )$
$\longrightarrow (g\ i\ j \models_i f)$
$i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i f)$
proof −
have *1*: $\exists\ ul.\ nlength\ ul = nlength\ \ g \wedge (\forall\ i.\ i \leq nlength\ g \longrightarrow (nnth\ ul\ i) = True)$
by (*metis inj-enat nlength-nmap nnth-nmap*)
obtain *ug* **where** *4*: $nlength\ ug = nlength\ g \wedge (\forall\ i.\ i \leq nlength\ g \longrightarrow (nnth\ ug\ i) = True)$
using *1* **by** *blast*
have *8*: $(g\ i\ j \models_i f) \longleftrightarrow\ ((upd\ g\ q\ ug)\ i\ j \models_i f)$
using *not-fvar-upd*[*of q f ug g i j*]
using *4 assms* **by** *blast*
show *?thesis* **using** *assms*
by (*simp add*: *4 8 nlength-upd nnth-upd*)
qed

lemma *Venema-Consistency*:
assumes $\vdash_i hor(q)\ iimp\ f$
$q \notin fvars\ f$
shows $\vdash_i f$
using *hor-rule*[*of q f*] *hor-defs assms* **by** *force*

definition *valid-finite* :: $cdt \Rightarrow bool\ ((\vdash_f \text{-})\ 5)$
where $(\vdash_f f) = (\forall\ g\ i\ j.\ (enat\ i) \leq j \wedge j \leq nlength\ g \wedge nfinite\ g \longrightarrow (g\ i\ j \models_i f))$

lemma *valid-finite*:
assumes $\vdash_i f$
shows $\vdash_f f$
using *assms valid-finite-def*
using *valid-def* **by** *blast*

primrec *rva* :: $cdt \Rightarrow cdt\ ((rva\ \text{-})\ [85]\ 85)$
where
$rva\ ifalse = ifalse$
$|\ rva\ (\$p) = (fin\ \$p)$
$|\ rva\ (f\ iimp\ g) = ((rva\ f)\ iimp\ (rva\ g))$
$|\ rva\ empty = empty$
$|\ rva\ (f\ cee\ g) = (\ (rva\ g)\ cee\ (rva\ f))$
$|\ rva\ (f\ tee\ g) = (\ (rva\ f)\ dee\ (rva\ g))$
$|\ rva\ (f\ dee\ g) = (\ (rva\ f)\ tee\ (rva\ g))$

lemma *rva-sem*:
assumes *nfinite g*
$i \leq j$
$j \leq$ *nlength g*
shows $(g\ i\ j \models_i (rva\ f)) = (\ (nrev\ g)\ (the\text{-}enat(nlength\ g)\ -j)\ (nlength\ g - i) \models_i f)$
using *assms*
proof (*induction f arbitrary*: *g i j*)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case* **apply** (*simp add*: *fin-defs*)
by (*metis diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *nfinite-conv-nlength-enat nnth-nrev the-enat.simps*)
next
case (*iimp-d f1 f2*)
then show *?case* **by** *simp*
next
case *empty-d*
then show *?case* **apply** (*simp*)
by (*metis diff-diff-cancel dual-order.trans enat-ord-simps*(*1*) *idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps*)
next
case (*c-d f1 f2*)
then show *?case*
proof −
have *c1*: $(g\ i\ j \models_i (rva\ (f1\ cee\ f2))) = (g\ i\ j \models_i (rva\ f2)\ cee\ (rva\ f1))$
by *simp*
have *c2*: $(g\ i\ j \models_i (rva\ f2)\ cee\ (rva\ f1)) =$
$(\exists k{\geq}i.\ k \leq j \wedge (g\ i\ (enat\ k) \models_i rva\ f2) \wedge (g\ k\ (enat\ j) \models_i rva\ f1))$
by *simp*
have *c3*: $((nrev\ g)\ (the\text{-}enat\ (nlength\ g) - j)\ (nlength\ g - enat\ i) \models_i\ f1\ cee\ f2\ ) =$
$(\exists k{\geq}the\text{-}enat\ (nlength\ g) - j.\ enat\ k \leq nlength\ g - enat\ i \wedge$
$((nrev\ g)\ (the\text{-}enat\ (nlength\ g) - j)\ (enat\ k) \models_i f1) \wedge ((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2))$
by *simp*
have *c4*: $(\exists k{\geq}i.\ k \leq j \wedge (g\ i\ (enat\ k) \models_i rva\ f2) \wedge (g\ k\ (enat\ j) \models_i rva\ f1)) \Longrightarrow$
$(\exists k{\geq}the\text{-}enat\ (nlength\ g) - j.\ enat\ k \leq nlength\ g - enat\ i \wedge$
$((nrev\ g)\ (the\text{-}enat\ (nlength\ g) - j)\ (enat\ k) \models_i f1) \wedge ((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2))$
proof −
assume *ca0*: $(\exists k{\geq}i.\ k \leq j \wedge (g\ i\ (enat\ k) \models_i rva\ f2) \wedge (g\ k\ (enat\ j) \models_i rva\ f1))$
show $(\exists k{\geq}the\text{-}enat\ (nlength\ g) - j.\ enat\ k \leq nlength\ g - enat\ i \wedge$
$((nrev\ g)\ (the\text{-}enat\ (nlength\ g) - j)\ (enat\ k) \models_i f1) \wedge ((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2))$
proof −

obtain *k* **where** *c100*: $k{\geq}i \wedge k \leq j \wedge (g\ i\ (enat\ k) \models_i rva\ f2) \wedge (g\ k\ (enat\ j) \models_i rva\ f1)$
using *ca0* **by** *blast*
have *c101*: $(\ (nrev\ g)\ (the\text{-}enat(nlength\ g) - k)\ (nlength\ g - i) \models_i f2)$
by (*meson c100 c-d.IH*(*2*) *c-d.prems*(*1*) *c-d.prems*(*3*) *enat-ord-simps*(*1*) *order-trans*)
have *c102*: $(\ (nrev\ g)\ (the\text{-}enat(nlength\ g) - j)\ (nlength\ g - k) \models_i f1)$
using *c100 c-d.IH*(*1*) *c-d.prems*(*1*) *c-d.prems*(*3*) **by** *blast*

show *?thesis*

by (*metis c100 c101 c102 c-d.prems(1) diff-le-mono2 enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps*)
qed
qed
have *c5*: (∃ *k*≥*the-enat* (*nlength g*) − *j*. *enat k* ≤ *nlength g* − *enat i* ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) (*enat k*) ⊨$_i$ *f1*) ∧ ((*nrev g*) *k* (*nlength g* − *enat i*) ⊨$_i$ *f2*))
⟹
(∃ *k*≥*i*. *k* ≤ *j* ∧ (*g i* (*enat k*) ⊨$_i$ *rva f2*) ∧ (*g k* (*enat j*) ⊨$_i$ *rva f1*))
proof −
assume *ca0*: (∃ *k*≥*the-enat* (*nlength g*) − *j*. *enat k* ≤ *nlength g* − *enat i* ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) (*enat k*) ⊨$_i$ *f1*) ∧ ((*nrev g*) *k* (*nlength g* − *enat i*) ⊨$_i$ *f2*))
show (∃ *k*≥*i*. *k* ≤ *j* ∧ (*g i* (*enat k*) ⊨$_i$ *rva f2*) ∧ (*g k* (*enat j*) ⊨$_i$ *rva f1*))
proof −
obtain *k* **where** *c200*: *k*≥*the-enat* (*nlength g*) − *j* ∧ *enat k* ≤ *nlength g* − *enat i* ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) (*enat k*) ⊨$_i$ *f1*) ∧ ((*nrev g*) *k* (*nlength g* − *enat i*) ⊨$_i$ *f2*)
using *ca0* **by** *blast*
have *c201*: *i* ≤ *the-enat* (*nlength g*) − *k*
by (*metis c200 c-d.prems(1) c-d.prems(2) c-d.prems(3) diff-diff-cancel diff-le-mono2 dual-order.trans enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps*)
have *c2011*: *the-enat* (*nlength g*) − (*the-enat* (*nlength g*) − *k*) = *k*
by (*metis c200 c201 c-d.prems(1) diff-diff-cancel diff-is-0-eq enat.distinct(2) enat-ord-simps(1) enat-the-enat idiff-0-right nfinite-conv-nlength-enat nle-le zero-enat-def*)
have *c202*: *nlength g* − *enat* (*the-enat* (*nlength g*) − *k*) = (*enat k*)
by (*metis c200 c-d.prems(1) diff-diff-cancel diff-le-self dual-order.trans enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps*)
have *c203*: *the-enat* (*nlength g*) − *k* ≤ *j*
using *c200* **by** *linarith*
have *c204*: (*g* (*the-enat* (*nlength g*) −*k*) (*enat j*) ⊨$_i$ *rva f1*) **using** *c-d.IH(1)*[*of g* (*the-enat* (*nlength g*) −*k*) *j*]
using *c200 c202 c203 c-d.prems(1) c-d.prems(3)* **by** *presburger*
have *c205*: (*g i* (*enat* (*the-enat* (*nlength g*) −*k*)) ⊨$_i$ (*rva f2*)) **using** *c-d.IH(2)*[*of g i* (*the-enat* (*nlength g*) −*k*)]
by (*metis c200 c201 c2011 c203 c-d.prems(1) c-d.prems(3) enat-ord-simps(1) order.trans*)
show *?thesis*
using *c201 c203 c204 c205* **by** *blast*
qed
qed
show *?thesis* **using** *c1 c2 c3 c4 c5* **by** *blast*
qed
next
case (*d-d f1 f2*)
then show *?case*
proof −
have *d1*: (*g i j* ⊨$_i$ *rva* (*f1 dee f2*)) =
(∃ *k*≥*enat j*. *k* ≤ *nlength g* ∧ (*g j k* ⊨$_i$ *rva f1*) ∧ (*g i k* ⊨$_i$ *rva f2*))
by *simp*
have *d2*: ((*nrev g*) (*the-enat* (*nlength g*) − *j*) (*nlength g* − *enat i*) ⊨$_i$ *f1 dee f2*) =
(∃ *k*≤*the-enat* (*nlength g*) − *j*. ((*nrev g*) *k* (*enat* (*the-enat* (*nlength g*) − *j*)) ⊨$_i$ *f1*) ∧
((*nrev g*) *k* (*nlength g* − *enat i*) ⊨$_i$ *f2*))
by *simp*

have *d3*: $(\exists k{\geq}enat\ j.\ k \leq nlength\ g \wedge (g\ j\ k \models_i rva\ f1) \wedge (g\ i\ k \models_i rva\ f2)) \Longrightarrow$
$(\exists k{\leq}the\text{-}enat\ (nlength\ g) - j.\ ((nrev\ g)\ k\ (enat\ (the\text{-}enat\ (nlength\ g) - j)) \models_i f1) \wedge$
$((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2))$
proof −
assume *da0*: $(\exists k{\geq}enat\ j.\ k \leq nlength\ g \wedge (g\ j\ k \models_i rva\ f1) \wedge (g\ i\ k \models_i rva\ f2))$
show $(\exists k{\leq}the\text{-}enat\ (nlength\ g) - j.\ ((nrev\ g)\ k\ (enat\ (the\text{-}enat\ (nlength\ g) - j)) \models_i f1) \wedge$
$((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2))$
proof −
obtain *k* **where** *d100*: $k{\geq}enat\ j \wedge k \leq nlength\ g \wedge (g\ j\ k \models_i rva\ f1) \wedge (g\ i\ k \models_i rva\ f2)$
using *da0* **by** *blast*
have *d103*: $(the\text{-}enat(nlength\ g - k)) \leq the\text{-}enat\ (nlength\ g) - j$
by (*metis d100 d-d.prems*(*1*) *diff-le-mono2 enat-ile enat-ord-simps*(*1*) *idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps*)
have *d101*: $((nrev\ g)\ (the\text{-}enat(nlength\ g - k))\ (enat\ (the\text{-}enat\ (nlength\ g) - j)) \models_i f1$
using *d-d.IH*(*1*)[*of g j -*] *d100*
by (*metis d-d.prems*(*1*) *enat-ile enat-ord-simps*(*1*) *idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps*)
have *d102*: $((nrev\ g)\ (the\text{-}enat(nlength\ g - k))\ (nlength\ g - enat\ i) \models_i f2)$
using *d100 d-d.IH*(*2*) *d-d.prems*(*1*) *d-d.prems*(*2*) *enat-ile nfinite-nlength-enat* **by** *fastforce*
show *?thesis*
using *d101 d102 d103* **by** *blast*
qed
qed
have *d4*: $(\exists k{\leq}the\text{-}enat\ (nlength\ g) - j.\ ((nrev\ g)\ k\ (enat\ (the\text{-}enat\ (nlength\ g) - j)) \models_i f1) \wedge$
$((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2)) \Longrightarrow$
$(\exists k{\geq}enat\ j.\ k \leq nlength\ g \wedge (g\ j\ k \models_i rva\ f1) \wedge (g\ i\ k \models_i rva\ f2))$
proof −
assume *da0*: $(\exists k{\leq}the\text{-}enat\ (nlength\ g) - j.\ ((nrev\ g)\ k\ (enat\ (the\text{-}enat\ (nlength\ g) - j)) \models_i f1) \wedge$
$((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2))$
show $(\exists k{\geq}enat\ j.\ k \leq nlength\ g \wedge (g\ j\ k \models_i rva\ f1) \wedge (g\ i\ k \models_i rva\ f2))$
proof −
obtain *k* **where** *d200*: $k{\leq}the\text{-}enat\ (nlength\ g) - j \wedge ((nrev\ g)\ k\ (enat\ (the\text{-}enat\ (nlength\ g) - j)) \models_i f1) \wedge$
$((nrev\ g)\ k\ (nlength\ g - enat\ i) \models_i f2)$
using *da0* **by** *blast*
have *d201*: $j \leq the\text{-}enat\ (nlength\ g) - k$
using *d200 d-d.prems*(*1*) *d-d.prems*(*3*) *nfinite-nlength-enat* **by** *fastforce*
have *d202*: $enat\ (the\text{-}enat\ (nlength\ g) - k) \leq nlength\ g$
by (*metis diff-le-self enat-ord-simps*(*1*) *enat-ord-simps*(*3*) *enat-the-enat*)
have *d203*: $the\text{-}enat\ (nlength\ g) - (the\text{-}enat\ (nlength\ g) - k) = k$
by (*meson d200 diff-diff-cancel diff-le-self dual-order.trans*)
have *d204*: $(g\ j\ (enat\ (the\text{-}enat(nlength\ g) - k)) \models_i rva\ f1)$
using *d-d.IH*(*1*)[*of g j* $(the\text{-}enat(nlength\ g) - k)$]
by (*metis d200 d201 d202 d-d.prems*(*1*) *diff-diff-cancel diff-is-0-eq′ enat.distinct*(*1*) *enat-the-enat idiff-enat-enat nfinite-conv-nlength-enat nle-le*)
have *d205*: $(g\ i\ (nlength\ g - k) \models_i rva\ f2)$
using *d-d.IH*(*2*)[*of g i* $(the\text{-}enat(nlength\ g) - k)$]
by (*metis d200 d201 d202 d203 d-d.prems*(*1*) *d-d.prems*(*2*) *dual-order.trans enat.distinct*(*1*) *enat-the-enat idiff-enat-enat nfinite-conv-nlength-enat*)
show *?thesis*

by (*metis d201 d202 d204 d205 d-d.prems(1) enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps*)
qed
qed
show *?thesis* **using** *d1 d2 d3 d4* **by** *blast*
qed
next
case (*t-d f1 f2*)
then show *?case*
proof −
have *t1*: (*g i j* ⊨$_i$ *rva* (*f1 tee f2*)) =
(∃ *k*≤*i*. (*g k* (*enat i*) ⊨$_i$ *rva f1*) ∧ (*g k* (*enat j*) ⊨$_i$ *rva f2*))
by *simp*
have *t2*: ((*nrev g*) (*the-enat* (*nlength g*) − *j*) (*nlength g* − *enat i*) ⊨$_i$ *f1 tee f2*) =
(∃ *k*≥*nlength g* − *enat i*. *k* ≤ *nlength g* ∧ ((*nrev g*) (*the-enat* (*nlength g* − *enat i*)) *k* ⊨$_i$ *f1*) ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) *k* ⊨$_i$ *f2*))
apply *simp*
apply *auto*
by (*metis enat-ord-code(5) enat-the-enat nfinite-conv-nlength-enat t-d.prems(1)*)

have *t3*: (∃ *k*≤*i*. (*g k* (*enat i*) ⊨$_i$ *rva f1*) ∧ (*g k* (*enat j*) ⊨$_i$ *rva f2*)) ⟹
(∃ *k*≥*nlength g* − *enat i*. *k* ≤ *nlength g* ∧ ((*nrev g*) (*the-enat* (*nlength g* − *enat i*)) *k* ⊨$_i$ *f1*) ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) *k* ⊨$_i$ *f2*))
proof −
assume *ta0*: (∃ *k*≤*i*. (*g k* (*enat i*) ⊨$_i$ *rva f1*) ∧ (*g k* (*enat j*) ⊨$_i$ *rva f2*))
show (∃ *k*≥*nlength g* − *enat i*. *k* ≤ *nlength g* ∧ ((*nrev g*) (*the-enat* (*nlength g* − *enat i*)) *k* ⊨$_i$ *f1*) ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) *k* ⊨$_i$ *f2*))
proof −
obtain *k* **where** *t100*: *k*≤*i* ∧ (*g k* (*enat i*) ⊨$_i$ *rva f1*) ∧ (*g k* (*enat j*) ⊨$_i$ *rva f2*)
using *ta0* **by** *blast*
have *t101*: ((*nrev g*) (*the-enat* (*nlength g* − *enat i*)) (*nlength g* − *k*) ⊨$_i$ *f1*)
by (*metis enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat order-subst2 t100 t-d.IH(1) t-d.prems(1) t-d.prems(2) t-d.prems(3) the-enat.simps*)
have *t102*: ((*nrev g*) (*the-enat* (*nlength g*) − *j*) (*nlength g* −*k*) ⊨$_i$ *f2*)
by (*meson order.trans t100 t-d.IH(2) t-d.prems(1) t-d.prems(2) t-d.prems(3)*)
have *t103*: (*nlength g* −*k*) ≥*nlength g* − *enat i* **using** *t100*
by (*metis diff-le-mono2 enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat t-d.prems(1)*)
have *t104*: (*nlength g* −*k*) ≤ *nlength g* **using** *t100*
by (*metis enat.distinct(2) enat-add-sub-same enat-le-plus-same(2) enat-minus-mono1*)
show *?thesis*
using *t101 t102 t103 t104* **by** *blast*
qed
qed
have *t4*: (∃ *k*≥*nlength g* − *enat i*. *k* ≤ *nlength g* ∧ ((*nrev g*) (*the-enat* (*nlength g* − *enat i*)) *k* ⊨$_i$ *f1*) ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) *k* ⊨$_i$ *f2*)) ⟹
(∃ *k*≤*i*. (*g k* (*enat i*) ⊨$_i$ *rva f1*) ∧ (*g k* (*enat j*) ⊨$_i$ *rva f2*))
proof −
assume *ta0*: (∃ *k*≥*nlength g* − *enat i*. *k* ≤ *nlength g* ∧ ((*nrev g*) (*the-enat* (*nlength g* − *enat i*)) *k* ⊨$_i$ *f1*) ∧
((*nrev g*) (*the-enat* (*nlength g*) − *j*) *k* ⊨$_i$ *f2*))

show ($\exists k{\leq}i.$ (g k (*enat* i) $\models_i$ *rva f1*) $\wedge$ (g k (*enat* j) $\models_i$ *rva f2*))
proof −
obtain k **where** *t200*: $k{\geq}$*nlength* g − *enat* $i \wedge k \leq$ *nlength* $g \wedge$ ((*nrev* g) (*the-enat* (*nlength* g − *enat* i)) $k \models_i$ *f1*) $\wedge$
((*nrev* g) (*the-enat* (*nlength* g) − j) $k \models_i$ *f2*)
using *ta0* **by** *blast*
have *t201*: (*the-enat*(*nlength* g) $-k$) $\leq i$ **using** *t200*
using *enat-ile nfinite-nlength-enat t-d.prems*(*1*) **by** *fastforce*
have *t202*: *nlength* g − *enat* (*the-enat* (*nlength* g − k)) = k
by (*metis add.commute enat-add-sub-same enat-ord-code*(*4*) *enat-the-enat leD le-iff-add t200*)
have *t203*: *the-enat* (*nlength* g) − i = (*the-enat* (*nlength* g − *enat* i))
by (*metis enat.distinct*(*2*) *enat.inject enat-the-enat idiff-enat-enat nfinite-conv-nlength-enat t-d.prems*(*1*))
have *t2031*: ((*nrev* g) (*the-enat* (*nlength* g) − i) (*nlength* g − *enat* (*the-enat* (*nlength* g − k))) $\models_i$ *f1*)
using *t200 t202 t203* **by** *presburger*
have *t2032*: *nfinite* $g \wedge$ *the-enat* (*nlength* g − k) $\leq i \wedge$ *enat* $i \leq$ *nlength* g
by (*metis enat-ile enat-ord-simps*(*1*) *nfinite-conv-nlength-enat order.trans t201 t-d.prems*(*1*) *t-d.prems*(*2*) *t-d.prems*(*3*) *the-enat.simps*)
have *t204*: (g (*the-enat*(*nlength* g $-k$)) (*enat* i) $\models_i$ *rva f1*)
using *t-d.IH*(*1*)[*of* g (*the-enat*(*nlength* g $-k$)) i] **using** *t200 t201 t202 t2031 t2032* **by** *blast*
have *t205*: (g (*the-enat*(*nlength* g $-k$)) (*enat* j) $\models_i$ *rva f2*)
using *t-d.IH*(*2*)[*of* g (*the-enat*(*nlength* g $-k$)) j]
using *t200 t202 t2032 t-d.prems*(*2*) *t-d.prems*(*3*) **by** *auto*
show *?thesis*
using *t201 t202 t203 t204 t205*
using *t2032* **by** *blast*
qed
qed
show *?thesis* **using** *t1 t2 t3 t4* **by** *blast*
qed
qed

lemma *rva-atom*:
assumes *nfinite* g
$i \leq j$
$j \leq$ *nlength* g
shows (g i $j \models_i$ (*rva* (\$$p$))) = ($g$ i $j \models_i$ *fin* (\$$p$))
using *assms* **by** *simp*

lemma *rva-fin-atom*:
assumes *nfinite* g
$i \leq j$
$j \leq$ *nlength* g
shows (g i $j \models_i$ (*rva* ((*fin* (\$$p$))))) = ($g$ i $j \models_i$ \$$p$)
using *assms* **unfolding** *fin-d-def diamond-d-def iand-d-def ior-d-def inot-d-def itrue-d-def*
apply *simp*
unfolding *fin-d-def diamond-d-def iand-d-def ior-d-def inot-d-def itrue-d-def*
by *simp*

lemma *rva-rva*:
assumes *nfinite* g

$i \leq j$
$j \leq nlength\ g$
shows $(g\ i\ j \models_i (rva\ (rva\ f))) = (g\ i\ j \models_i f)$
using *assms*
proof (*induction f arbitrary*: $g\ i\ j$)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case* **apply** *simp*
using *rva-fin-atom*
using *semantics-cdt.simps*(*2*) **by** *presburger*
next
case (*iimp-d f1 f2*)
then show *?case* **by** *simp*
next
case *empty-d*
then show *?case* **by** *simp*
next
case (*c-d f1 f2*)
then show *?case* **apply** *simp*
by (*meson dual-order.trans enat-ord-simps*(*1*))
next
case (*d-d f1 f2*)
then show *?case* **apply** *simp*
by (*meson dual-order.trans enat-ord-simps*(*1*))
next
case (*t-d f1 f2*)
then show *?case* **apply** *simp*
by (*metis* (*no-types*, *lifting*) *dual-order.trans the-enat.simps*)
qed

lemma *rva-valid-1*:
assumes $\bigwedge g\ i\ j.\ enat\ i \leq j \wedge j \leq nlength\ g \wedge nfinite\ g \Longrightarrow (g\ i\ j \models_i f)$
$enat\ i \leq j$
$enat\ j \leq nlength\ g$
$nfinite\ g$
shows $(g\ i\ j \models_i rva\ f)$
using *assms* **using** *rva-sem*[*of g i j f*]
by (*metis diff-diff-left diff-le-self enat-ord-simps*(*1*) *idiff-enat-enat le-iff-add nfinite-conv-nlength-enat nlength-nrev the-enat.simps*)

lemma *rva-valid-2*:
assumes $\bigwedge g\ i\ j.\ enat\ i \leq j \wedge j \leq nlength\ g \wedge nfinite\ g \Longrightarrow (g\ i\ j \models_i rva\ f)$
$enat\ i \leq j$
$enat\ j \leq nlength\ g$
$nfinite\ g$
shows $(g\ i\ j \models_i f)$

using *assms rva-sem*[*of*]
by *simp* (*meson assms*(*2*) *rva-rva rva-valid-1*)

lemma *rva-valid*:
$(\vdash_f f) = (\vdash_f rva\ f)$
unfolding *valid-finite-def*
by (*metis enat-ile nfinite-nlength-enat rva-valid-1 rva-valid-2*)

end

Chapter 7

NL-ITL

theory *NL-ITL-Deep* **imports**
Extended-Int NELList-Extras

begin

This theory contains the syntax and semantics of ITL with neighbourhood operators, i.e., NL-ITL[5]. Note, we have both infinite past and infinite future, therefore we need the theory of extended integers.

7.1 Syntax

7.1.1 Primitive formulae

datatype *nl-itl* =
false-d (*ifalse*)
| *atom-d nat* (($-) [*100*] *99*)
| *iimp-d nl-itl nl-itl* ((- *iimp* -) [*26*,*25*] *25*)
| *next-d nl-itl* ((*next* -) [*88*] *87*)
| *chop-d nl-itl nl-itl* ((- *schop* -) [*84*,*84*] *83*)
| *ldiamond-d nl-itl* ((*ldiamond* -) [*88*] *87*)
| *rdiamond-d nl-itl* ((*rdiamond* -) [*88*] *87*)
| *chopstar-d nl-itl* ((*chopstar* -) [*85*] *85*)
| *exists-d nat nl-itl* (*Ex* -. - *10*)

datatype *introspective-nl-itl* =
cfalse-d (*cifalse*)
| *catom-d nat* (($$-) [*100*] *99*)
| *ciimp-d introspective-nl-itl introspective-nl-itl* ((- *ciimp* -) [*26*,*25*] *25*)
| *cnext-d introspective-nl-itl* ((*cnext* -) [*88*] *87*)
| *cchop-d introspective-nl-itl introspective-nl-itl* ((- *cschop* -) [*84*,*84*] *83*)
| *cchopstar-d introspective-nl-itl* ((*cchopstar* -) [*85*] *85*)
| *cexists-d nat introspective-nl-itl* (*cEx* -. - *10*)

datatype *future-nl-itl* =
fintro-d introspective-nl-itl ((*fintro* -) [*88*] *87*)

| *finot-d future-nl-itl* ((*finot* -) [*90*] *90*)
| *fior-d future-nl-itl future-nl-itl* ((- *fior* -) [*31*,*30*] *30*)
| *frdiamond-d future-nl-itl* ((*frdiamond* -) [*88*] *87*)

datatype *stricly-future-nl-itl* =
sfrdiamond-d future-nl-itl ((*sfrdiamond* -) [*88*] *87*)

datatype *past-nl-itl* =
pintro-d introspective-nl-itl ((*pintro* -) [*88*] *87*)
| *pinot-d past-nl-itl* ((*pinot* -) [*90*] *90*)
| *pior-d past-nl-itl past-nl-itl* ((- *pior* -) [*31*,*30*] *30*)
| *pldiamond-d past-nl-itl* ((*pldiamond* -) [*88*] *87*)

datatype *strictly-past-nl-itl* =
spldiamond-d past-nl-itl ((*spldiamond* -) [*88*] *87*)

datatype *separated-nl-itl* =
past-d past-nl-itl ((*past* -) [*88*] *87*)
| *future-d future-nl-itl* ((*future* -) [*88*] *87*)
| *sinot-d separated-nl-itl* ((*sinot* -) [*90*] *90*)
| *sior-d separated-nl-itl separated-nl-itl* ((- *sior* -) [*31*,*30*] *30*)

7.1.2 state, introspective, future and past formula

fun *state-nl-itl* :: *nl-itl* ⇒ *bool*
where
state-nl-itl (*ifalse*) = *True*
| *state-nl-itl* (\$*n*) = *True*
| *state-nl-itl* (*f iimp g*) = ((*state-nl-itl f*) ∧ (*state-nl-itl g*))
| *state-nl-itl* (*next f*) = *False*
| *state-nl-itl* (*f schop g*) = *False*
| *state-nl-itl* (*ldiamond f*) = *False*
| *state-nl-itl* (*rdiamond f*) = *False*
| *state-nl-itl* (*chopstar f*) = *False*
| *state-nl-itl* (*Ex n. f*) = *state-nl-itl f*

fun *introspective-nl-itl-prop* :: *nl-itl* ⇒ *bool*
where
introspective-nl-itl-prop (*ifalse*) = *True*
| *introspective-nl-itl-prop* (\$*n*) = *True*
| *introspective-nl-itl-prop* (*f iimp g*) = ((*introspective-nl-itl-prop f*) ∧ (*introspective-nl-itl-prop g*))
| *introspective-nl-itl-prop* (*next f*) = *introspective-nl-itl-prop f*
| *introspective-nl-itl-prop* (*f schop g*) = ((*introspective-nl-itl-prop f*) ∧ (*introspective-nl-itl-prop g*))
| *introspective-nl-itl-prop* (*ldiamond f*) = *False*
| *introspective-nl-itl-prop* (*rdiamond f*) = *False*
| *introspective-nl-itl-prop* (*chopstar f*) = *introspective-nl-itl-prop f*
| *introspective-nl-itl-prop* (*Ex n. f*) = *introspective-nl-itl-prop f*

fun *future-nl-itl-prop* :: *nl-itl* ⇒ *bool*
where
future-nl-itl-prop (*ifalse*) = *True*
| *future-nl-itl-prop* ($*n*) = *True*
| *future-nl-itl-prop* (*f iimp g*) = ((*future-nl-itl-prop f*) ∧ (*future-nl-itl-prop g*))
| *future-nl-itl-prop* (*next f*) = *future-nl-itl-prop f*
| *future-nl-itl-prop* (*f schop g*) = ((*future-nl-itl-prop f*) ∧ (*future-nl-itl-prop g*))
| *future-nl-itl-prop* (*ldiamond f*) = *False*
| *future-nl-itl-prop* (*rdiamond f*) = *future-nl-itl-prop f*
| *future-nl-itl-prop* (*chopstar f*) = *future-nl-itl-prop f*
| *future-nl-itl-prop* (*Ex n. f*) = *future-nl-itl-prop f*

fun *past-nl-itl-prop* :: *nl-itl* ⇒ *bool*
where
past-nl-itl-prop (*ifalse*) = *True*
| *past-nl-itl-prop* ($*n*) = *True*
| *past-nl-itl-prop* (*f iimp g*) = ((*past-nl-itl-prop f*) ∧ (*past-nl-itl-prop g*))
| *past-nl-itl-prop* (*next f*) = *past-nl-itl-prop f*
| *past-nl-itl-prop* (*f schop g*) = ((*past-nl-itl-prop f*) ∧ (*past-nl-itl-prop g*))
| *past-nl-itl-prop* (*ldiamond f*) = (*past-nl-itl-prop f*)
| *past-nl-itl-prop* (*rdiamond f*) = *False*
| *past-nl-itl-prop* (*chopstar f*) = *past-nl-itl-prop f*
| *past-nl-itl-prop* (*Ex n. f*) = *past-nl-itl-prop f*

7.1.3 Derived Operators

definition *inot-d* ((*inot* -) [*90*] *90*)
where
inot f ≡ *f iimp ifalse*

definition *cinot-d* ((*cinot* -) [*90*] *90*)
where
cinot f ≡ *f ciimp cifalse*

definition *itrue-d* (*itrue*)
where
itrue ≡ *inot ifalse*

definition *citrue-d* (*citrue*)
where
citrue ≡ *cinot cifalse*

definition *ior-d* ((- *ior* -) [*31*,*30*] *30*)
where
f ior g ≡ (*inot f*) *iimp g*

definition *cior-d* ((- *cior* -) [*31*,*30*] *30*)

where
f cior g ≡ (*cinot f*) *ciimp g*

definition *iand-d* ((- *iand* -) [*36*,*35*] *35*)
where
f iand g ≡ *inot* (*inot f ior inot g*)

definition *ciand-d* ((- *ciand* -) [*36*,*35*] *35*)
where
f ciand g ≡ *cinot* (*cinot f cior cinot g*)

definition *ieqv-d* ((- *ieqv* -) [*21*,*20*] *20*)
where
f ieqv g ≡ ((*f iimp g*) *iand* (*g iimp f*))

definition *cieqv-d* ((- *cieqv* -) [*21*,*20*] *20*)
where
f cieqv g ≡ ((*f ciimp g*) *ciand* (*g ciimp f*))

7.1.4 more, empty, skip and wnext

definition *more-d* (*more*)
where
more ≡ *itrue schop* (*next itrue*)

definition *cmore-d* (*cmore*)
where
cmore ≡ *citrue cschop* (*cnext citrue*)

definition *empty-d* (*empty*)
where
empty ≡ *inot more*

definition *cempty-d* (*cempty*)
where
cempty ≡ *cinot cmore*

definition *skip-d* (*skip*)
where
skip ≡ *next empty*

definition *cskip-d* (*cskip*)
where
cskip ≡ *cnext cempty*

definition *wnext-d* ((*wnext* -) [*88*] *87*)
where
wnext f ≡ *inot* (*next* (*inot f*))

definition *cwnext-d* ((*cwnext* -) [*88*] *87*)
where
cwnext f ≡ *cinot* (*cnext* (*cinot f*))

definition *prev-d* ((*prev* -) [*88*] *87*)
where
prev f ≡ *f schop skip*

definition *cprev-d* ((*cprev* -) [*88*] *87*)
where
cprev f ≡ *f cschop cskip*

definition *wprev-d* ((*wprev* -) [*88*] *87*)
where
wprev f ≡ *inot*(*prev* (*inot f*))

definition *cwprev-d* ((*cwprev* -) [*88*] *87*)
where
cwprev f ≡ *cinot*(*cprev* (*cinot f*))

7.1.5 rfinite, lfinite, rinf and linf

definition *rfinite-d* (*rfinite*)
where
rfinite ≡ *itrue schop empty*

definition *crfinite-d* (*crfinite*)
where
crfinite ≡ *citrue cschop cempty*

definition *rinf-d* (*rinf*)
where
rinf ≡ *inot rfinite*

definition *crinf-d* (*crinf*)
where
crinf ≡ *cinot crfinite*

definition *lfinite-d* (*lfinite*)
where
lfinite ≡ (*next itrue*) *ior empty*

definition *clfinite-d* (*clfinite*)
where
clfinite ≡ (*cnext citrue*) *cior cempty*

definition *linf-d* (*linf*)
where

linf ≡ *inot lfinite*

definition *clinf-d* (*clinf*)
where
clinf ≡ *cinot clfinite*

7.1.6 weak chop

definition *wchop-d* ((- *wchop* -) [*84*,*84*] *83*)
where *f wchop g* ≡ (*f schop g*) *ior* (*f iand rinf*)

definition *cwchop-d* ((- *cwchop* -) [*84*,*84*] *83*)
where *f cwchop g* ≡ (*f cschop g*) *cior* (*f ciand crinf*)

7.1.7 Box and Diamond Operators

definition *sometimes-d* ((*diamond* -) [*88*] *87*)
where
diamond f ≡ *itrue schop f*

definition *csometimes-d* ((*cdiamond* -) [*88*] *87*)
where
cdiamond f ≡ *citrue cschop f*

definition *always-d* ((*box* -) [*88*] *87*)
where
box f ≡ *inot* (*diamond* (*inot f*))

definition *calways-d* ((*cbox* -) [*88*] *87*)
where
cbox f ≡ *cinot* (*cdiamond* (*cinot f*))

definition *di-d* ((*di* -) [*88*] *87*)
where
di f ≡ *f schop itrue*

definition *cdi-d* ((*cdi* -) [*88*] *87*)
where
cdi f ≡ *f cschop citrue*

definition *bi-d* ((*bi* -) [*88*] *87*)
where
bi f ≡ *inot* (*di* (*inot f*))

definition *cbi-d* ((*cbi* -) [*88*] *87*)
where
cbi f ≡ *cinot* (*cdi* (*cinot f*))

7.1.8 Initial and Final Operators

definition *init-d* ((*init* -) [*88*] *87*)
where
init f ≡ (*empty iand f*) *schop itrue*

definition *cinit-d* ((*cinit* -) [*88*] *87*)
where
cinit f ≡ (*cempty ciand f*) *cschop citrue*

definition *fin-d* ((*fin* -) [*88*] *87*)
where
fin f ≡ *itrue schop* (*empty iand f*)

definition *cfin-d* ((*cfin* -) [*88*] *87*)
where
cfin f ≡ *citrue cschop* (*cempty ciand f*)

7.1.9 Forall

definition *forall-d* (*Fa* -. - *10*)
where
Fa v. f ≡ *inot* (*Ex v. inot f*)

definition *cforall-d* (*cFa* -. - *10*)
where
cFa v. f ≡ *cinot* (*cEx v. cinot f*)

7.1.10 Big operations

definition *big-ior* :: *nl-itl list* ⇒ *nl-itl*
where *big-ior F* = *foldr* (λ *x y. x ior y*) *F ifalse*

definition *big-cior* :: *introspective-nl-itl list* ⇒ *introspective-nl-itl*
where *big-cior F* = *foldr* (λ *x y. x cior y*) *F cifalse*

definition *big-iand* :: *nl-itl list* ⇒ *nl-itl*
where *big-iand F* = *foldr* (λ *x y. x iand y*) *F itrue*

definition *big-ciand* :: *introspective-nl-itl list* ⇒ *introspective-nl-itl*
where *big-ciand F* = *foldr* (λ *x y. x ciand y*) *F citrue*

definition *state-nl-itl-list* :: *nl-itl list* ⇒ *bool*
where *state-nl-itl-list L* = *foldr* (λ*x y. state-nl-itl x* ∧ *y*) *L True*

7.2 Semantics

7.2.1 Intervals

type-synonym *interval* = *int* ⇒ *nat set*

7.2.2 Semantics Primitive Operators

definition *similar* :: *interval* ⇒ *nat* ⇒ *interval* ⇒ *bool* ((- ∼- -))
where *similar* $\sigma 0\ n\ \sigma 1 = (\forall\ k\ p\ .\ (p \in (\sigma 0\ k) \wedge p \neq n) \longleftrightarrow (p \in (\sigma 1\ k) \wedge p \neq n)\ )$

definition *csimilar* :: *interval* ⇒ *nat* ⇒ *eint* ⇒ *eint* ⇒ *interval* ⇒ *bool*
where *csimilar* $\sigma 0\ n\ i\ j\ \sigma 1 =$
$(\forall\ k\ .\ \mathit{if}\ i \leq k \wedge k \leq j\ \mathit{then}\ (\forall p.\ (p \in (\sigma 0\ k) \wedge p \neq n) \longleftrightarrow (p \in (\sigma 1\ k) \wedge p \neq n))$
$\mathit{else}\ (\forall\ p.\ (p \in (\sigma 0\ k)) \longleftrightarrow (p \in (\sigma 1\ k)))\ )$

definition *similars* :: *interval* ⇒ *nat set* ⇒ *interval* ⇒ *bool* ((- ≈- -))
where *similars* $\sigma 0\ A\ \sigma 1 = (\forall\ k\ p.\ p \notin A \longrightarrow (p \in (\sigma 0\ k) \longleftrightarrow p \in (\sigma 1\ k)))$

definition *upd-add-sigma* :: *interval* ⇒ *int* ⇒ *nat* ⇒ *interval*
where *upd-add-sigma* $\sigma\ k\ n = \sigma(k := \mathit{insert}\ n\ (\sigma\ k))$

definition *upd-remove-sigma* :: *interval* ⇒ *int* ⇒ *nat set* ⇒ *interval*
where *upd-remove-sigma* $\sigma\ k\ A = \sigma(k :=\ (\sigma\ k) - A\ )$

definition *upd-remove-all-sigma* :: *interval* ⇒ *nat set* ⇒ *interval*
where *upd-remove-all-sigma* $\sigma\ A = (\lambda\ i.\ (\mathit{upd\text{-}remove\text{-}sigma}\ \sigma\ i\ A)\ i)$

primrec *fvars* :: *nl-itl* ⇒ *nat set*
where
fvars ifalse $= \{\}$
| *fvars* ($\$n$) $= \{n\}$
| *fvars* (*f iimp g*) $= (\mathit{fvars}\ f) \cup (\mathit{fvars}\ g)$
| *fvars* (*next f*) $= \mathit{fvars}\ f$
| *fvars* (*f schop g*) $= (\mathit{fvars}\ f) \cup (\mathit{fvars}\ g)$
| *fvars* (*ldiamond f*) $= \mathit{fvars}\ f$
| *fvars* (*rdiamond f*) $= \mathit{fvars}\ f$
| *fvars* (*chopstar f*) $= \mathit{fvars}\ f$
| *fvars* (*Ex n. f*) $= (\mathit{fvars}\ f) - \{n\}$

primrec *cfvars* :: *introspective-nl-itl* ⇒ *nat set*
where
cfvars cifalse $= \{\}$
| *cfvars* ($\$\$n$) $= \{n\}$
| *cfvars* (*f ciimp g*) $= (\mathit{cfvars}\ f) \cup (\mathit{cfvars}\ g)$
| *cfvars* (*cnext f*) $= \mathit{cfvars}\ f$
| *cfvars* (*f cschop g*) $= (\mathit{cfvars}\ f) \cup (\mathit{cfvars}\ g)$
| *cfvars* (*cchopstar f*) $= \mathit{cfvars}\ f$
| *cfvars* (*cEx n. f*) $= (\mathit{cfvars}\ f) - \{n\}$

primrec *bvars* :: *nl-itl* ⇒ *nat set*

where
  bvars ifalse = {}
| *bvars* (\$*n*) = {}
| *bvars* (*f iimp g*) = (*bvars f*) ∪ (*bvars g*)
| *bvars* (*next f*) = *bvars f*
| *bvars* (*f schop g*) = (*bvars f*) ∪ (*bvars g*)
| *bvars* (*ldiamond f*) = *bvars f*
| *bvars* (*rdiamond f*) = *bvars f*
| *bvars* (*chopstar f*) = *bvars f*
| *bvars* (*Ex n. f*) = (*insert n* (*bvars f*))

primrec *cbvars* :: *introspective-nl-itl* ⇒ *nat set*
where
  cbvars cifalse = {}
| *cbvars* (\$\$*n*) = {}
| *cbvars* (*f ciimp g*) = (*cbvars f*) ∪ (*cbvars g*)
| *cbvars* (*cnext f*) = *cbvars f*
| *cbvars* (*f cschop g*) = (*cbvars f*) ∪ (*cbvars g*)
| *cbvars* (*cchopstar f*) = *cbvars f*
| *cbvars* (*cEx n. f*) = (*insert n* (*cbvars f*))

primrec *suform* :: *nl-itl* ⇒ *nat* ⇒ *nl-itl* ⇒ *nl-itl*
where
  suform ifalse n h = *ifalse*
| *suform* (\$*k*) *n h* = (*if k* = *n then h else* (\$*k*))
| *suform* (*f iimp g*) *n h* = ((*suform f n h*) *iimp* (*suform g n h*))
| *suform* (*next f*) *n h* = (*next* (*suform f n h*))
| *suform* (*f schop g*) *n h* = ((*suform f n h*) *schop* (*suform g n h*))
| *suform* (*ldiamond f*) *n h* = (*ldiamond* (*suform f n h*))
| *suform* (*rdiamond f*) *n h* = (*rdiamond* (*suform f n h*))
| *suform* (*chopstar f*) *n h* = (*chopstar* (*suform f n h*))
| *suform* (*Ex k. f*) *n h* = (*Ex k.* (*suform f n h*))

primrec *csuform* :: *introspective-nl-itl* ⇒ *nat* ⇒ *introspective-nl-itl* ⇒ *introspective-nl-itl*
where
  csuform cifalse n h = *cifalse*
| *csuform* (\$\$*k*) *n h* = (*if k* = *n then h else* (\$\$*k*))
| *csuform* (*f ciimp g*) *n h* = ((*csuform f n h*) *ciimp* (*csuform g n h*))
| *csuform* (*cnext f*) *n h* = (*cnext* (*csuform f n h*))
| *csuform* (*f cschop g*) *n h* = ((*csuform f n h*) *cschop* (*csuform g n h*))
| *csuform* (*cchopstar f*) *n h* = (*cchopstar* (*csuform f n h*))
| *csuform* (*cEx k. f*) *n h* = (*cEx k.* (*csuform f n h*))

primrec *crename* :: *introspective-nl-itl* ⇒ *nat* ⇒ *nat* ⇒ *introspective-nl-itl*
where
  crename cifalse n k = *cifalse*
| *crename* (\$\$*m*) *n k* = (*if m* = *n then* \$\$*k else* \$\$*m*)
| *crename* (*f ciimp g*) *n k* = ((*crename f n k*) *ciimp* (*crename g n k*))

| *crename* (*cnext f*) *n k* = (*cnext* (*crename f n k*))
| *crename* (*f cschop g*) *n k* = ((*crename f n k*) *cschop* (*crename g n k*))
| *crename* (*cchopstar f*) *n k* = (*cchopstar* (*crename f n k*))
| *crename* (*cEx m. f*) *n k* = (*cEx* (*if m = n then k else m*). (*crename f n k*))

primrec *rename* :: *nl-itl* ⇒ *nat* ⇒ *nat* ⇒ *nl-itl*
where
rename ifalse n k = *ifalse*
| *rename* (*$m*) *n k* = (*if m = n then $k else $m*)
| *rename* (*f iimp g*) *n k* = ((*rename f n k*) *iimp* (*rename g n k*))
| *rename* (*next f*) *n k* = (*next* (*rename f n k*))
| *rename* (*f schop g*) *n k* = ((*rename f n k*) *schop* (*rename g n k*))
| *rename* (*ldiamond f*) *n k* = (*ldiamond* (*rename f n k*))
| *rename* (*rdiamond f*) *n k* = (*rdiamond* (*rename f n k*))
| *rename* (*chopstar f*) *n k* = (*chopstar* (*rename f n k*))
| *rename* (*Ex m. f*) *n k* = (*Ex* (*if m = n then k else m*). (*rename f n k*))

primrec *max-var* :: *nl-itl* ⇒ *nat* ⇒ *nat*
where
max-var ifalse m = *m*
| *max-var* (*$k*) *m* = (*if m < k then k else m*)
| *max-var* (*f iimp g*) *m* = (*if* (*max-var f m*) < (*max-var g m*) *then* (*max-var g m*) *else* (*max-var f m*))
| *max-var* (*next f*) *m* = *max-var f m*
| *max-var* (*f schop g*) *m* = (*if* (*max-var f m*) < (*max-var g m*) *then* (*max-var g m*) *else* (*max-var f m*))
| *max-var* (*ldiamond f*) *m* = *max-var f m*
| *max-var* (*rdiamond f*) *m* = *max-var f m*
| *max-var* (*chopstar f*) *m* = *max-var f m*
| *max-var* (*Ex k. f*) *m* = (*if* (*max-var f m*) < *k then k else* (*max-var f m*))

primrec *cmax-var* :: *introspective-nl-itl* ⇒ *nat* ⇒ *nat*
where
cmax-var cifalse m = *m*
| *cmax-var* (*$$k*) *m* = (*if m < k then k else m*)
| *cmax-var* (*f ciimp g*) *m* = (*if* (*cmax-var f m*) < (*cmax-var g m*) *then* (*cmax-var g m*) *else* (*cmax-var f m*))
| *cmax-var* (*cnext f*) *m* = *cmax-var f m*
| *cmax-var* (*f cschop g*) *m* = (*if* (*cmax-var f m*) < (*cmax-var g m*) *then* (*cmax-var g m*) *else* (*cmax-var f m*))
| *cmax-var* (*cchopstar f*) *m* = *cmax-var f m*
| *cmax-var* (*cEx k. f*) *m* = (*if* (*cmax-var f m*) < *k then k else* (*cmax-var f m*))

definition *upd* :: *interval* ⇒ *nat* ⇒ (*int* ⇒ *bool*) ⇒ *interval*
where *upd σ n l* = (λ *k. if* (*l k*) *then insert n* (*σ k*) *else* ((*σ k*) − {*n*}))

fun *semantics-nl-itl* :: *interval* ⇒ *eint* ⇒ *eint* ⇒ *nl-itl* ⇒ *bool*
((- - - ⊨ -) [80,80,80,10] 10)
where
(σ i j ⊨ *ifalse*) = *False*
| (σ i j ⊨ ($p)) = (i ≠ −∞ ∧ (p ∈ (σ (int-of-eint i))))
| (σ i j ⊨ (f iimp g)) = ((σ i j ⊨ f) ⟶ (σ i j ⊨ g))
| (σ i j ⊨ (next f)) = (i ≠ −∞ ∧ i < j ∧ (σ (i+ (eint 1)) j ⊨ f))
| (σ i j ⊨ (f schop g)) = (∃ k. i ≤ eint k ∧ eint k ≤ j ∧ (σ i k ⊨ f) ∧ (σ k j ⊨ g))
| (σ i j ⊨ ldiamond f) = (i > −∞ ∧ (∃ k. k ≤ i ∧ (σ k i ⊨ f)))
| (σ i j ⊨ rdiamond f) = (j < ∞ ∧ (∃ k. k ≥ j ∧ (σ j k ⊨ f)))
| (σ i j ⊨ (chopstar f)) =
(i = j ∨
(i≠j ∧ i ≠ −∞ ∧ j ≠ ∞ ∧
(∃ (lrf:: eint nellist).
(nnth lrf 0) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
nridx (λ a b. a < b ∧(σ a b ⊨ f)) lrf
)
) ∨
(i≠j ∧ i = −∞ ∧ j ≠ ∞ ∧
(∃ (li :: eint nellist).
(nnth li 0) = j ∧ ¬ nfinite li ∧
nridx (λ a b. b < a ∧(σ b a ⊨ f)) li
)
) ∨
(i≠j ∧ i ≠ −∞ ∧ j = ∞ ∧
(∃ (ri :: eint nellist).
(nnth ri 0) = i ∧ ¬ nfinite ri ∧
nridx (λ a b. a < b ∧ (σ a b ⊨ f)) ri
)
) ∨
(i≠j ∧ i = −∞ ∧ j = ∞ ∧
(∃ (li :: int nellist) (ri :: int nellist) .
(nnth li 0) = (nnth ri 0) ∧
¬ nfinite li ∧ nridx (λ a b. b < a ∧ (σ b a ⊨ f)) li ∧
¬ nfinite ri ∧ nridx (λ a b. a < b ∧ (σ a b ⊨ f)) ri
)
)
)

| (σ i j ⊨ Ex n. f) = (∃ l. ((upd σ n l) i j ⊨ f))

fun *semantics-introspective-nl-itl* :: *interval* ⇒ *eint* ⇒ *eint* ⇒ *introspective-nl-itl* ⇒ *bool*
((- - - c⊨ -) [80,80,80,10] 10)
where
(σ i j c⊨ *cifalse*) = *False*
| (σ i j c⊨ ($$p)) = (i ≠ −∞ ∧ (p ∈ (σ (int-of-eint i))))
| (σ i j c⊨ (f ciimp g)) = ((σ i j c⊨ f) ⟶ (σ i j c⊨ g))

$$
\begin{array}{l}
\mid (\ \sigma\ i\ j\ c\models (\mathit{cnext}\ f)) \quad = (\ i \neq -\infty \wedge i < j \wedge (\ \sigma\ (i+\ (\mathit{eint}\ 1)\ )\ j\ c\models f)) \\
\mid (\ \sigma\ i\ j\ c\models (f\ \mathit{cschop}\ g)) = (\exists\, k.\ \ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge\ (\ \sigma\ i\ k\ c\models\ f) \wedge (\ \sigma\ k\ j\ c\models g)) \\
\mid (\ \sigma\ i\ j\ c\models (\mathit{cchopstar}\ f)) = \\
\quad (i = j \vee \\
\quad (\ i{\neq}j \wedge i \neq -\infty \wedge j \neq \infty \wedge \\
\quad\ (\exists\ (\mathit{lrf}{::}\ \mathit{eint}\ \mathit{nellist}). \\
\quad\ (\mathit{nnth}\ \mathit{lrf}\ 0) = i \wedge \mathit{nfinite}\ \mathit{lrf} \wedge \mathit{nlast}\ \mathit{lrf} = j \wedge \\
\quad\ \mathit{nridx}\ (\lambda\ a\ b.\ a < b \wedge (\sigma\ a\ b\ c\models f))\ \mathit{lrf} \\
\quad\ ) \\
\quad\ ) \vee \\
\quad (\ i{\neq}j \wedge i = -\infty \wedge j \neq \infty \wedge \\
\quad\ (\exists\ (\mathit{li}\ ::\ \mathit{eint}\ \mathit{nellist}). \\
\quad\ (\mathit{nnth}\ \mathit{li}\ 0) = j \wedge \neg\ \mathit{nfinite}\ \mathit{li}\ \wedge \\
\quad\ \mathit{nridx}\ (\lambda\ a\ b.\ b < a \wedge (\sigma\ b\ a\ c\models f))\ \mathit{li} \\
\quad\ ) \\
\quad\ ) \vee \\
\quad (\ i{\neq}j \wedge i \neq -\infty \wedge j = \infty \wedge \\
\quad\ (\exists\ (\mathit{ri}\ ::\ \mathit{eint}\ \mathit{nellist}). \\
\quad\ (\mathit{nnth}\ \mathit{ri}\ 0) = i \wedge \neg\ \mathit{nfinite}\ \mathit{ri}\ \wedge \\
\quad\ \mathit{nridx}\ (\lambda\ a\ b.\ a < b \wedge (\sigma\ a\ b\ c\models f))\ \mathit{ri} \\
\quad\ ) \\
\quad\ ) \vee \\
\quad (\ i{\neq}j \wedge i = -\infty \wedge j = \infty \wedge \\
\quad\ (\exists\ (\mathit{li}\ ::\ \mathit{int}\ \mathit{nellist})\ (\mathit{ri}\ ::\ \mathit{int}\ \mathit{nellist})\ . \\
\quad\ \ (\mathit{nnth}\ \mathit{li}\ 0) = (\mathit{nnth}\ \mathit{ri}\ 0) \wedge \\
\quad\ \neg\ \mathit{nfinite}\ \mathit{li}\ \wedge \mathit{nridx}\ (\lambda\ a\ b.\ b < a \wedge (\sigma\ b\ a\ c\models f))\ \mathit{li} \wedge \\
\quad\ \neg\ \mathit{nfinite}\ \mathit{ri}\ \wedge \mathit{nridx}\ (\lambda\ a\ b.\ a < b \wedge (\sigma\ a\ b\ c\models f))\ \mathit{ri} \\
\quad\ ) \\
\quad\ ) \\
\quad\ ) \\
\\
\mid\ (\sigma\ i\ j\ c\models \mathit{cEx}\ n.\ f) = (\exists\ l.\ ((\mathit{upd}\ \sigma\ n\ l)\ i\ j\ c\models f)\ \ )
\end{array}
$$

fun *semantics-future-nl-itl* :: *interval* ⇒ *eint* ⇒ *eint* ⇒ *future-nl-itl* ⇒ *bool*
((- - - *f*⊨ -) [*80*,*80*,*80*,*10*] *10*)
where

$$
\begin{array}{l}
\ (\ \sigma\ i\ j\ f\models \mathit{fintro}\ f) \quad = \ (\sigma\ i\ j\ c\models f) \\
\mid (\ \sigma\ i\ j\ f\models \mathit{finot}\ f) \quad = (\neg\ (\sigma\ i\ j\ f\models f)\ ) \\
\mid (\ \sigma\ i\ j\ f\models (f\ \mathit{fior}\ g)) \quad = (\ (\ \sigma\ i\ j\ f\models\ f) \vee (\ \sigma\ i\ j\ f\models g)\ ) \\
\mid (\ \sigma\ i\ j\ f\models \mathit{frdiamond}\ f) \ = (\ j <\ \infty \wedge (\exists\, k.\ \ k \geq j \wedge (\sigma\ j\ k\ f\models f)))
\end{array}
$$

fun *semantics-strictly-future-nl-itl* :: *interval* ⇒ *eint* ⇒ *eint* ⇒ *stricly-future-nl-itl* ⇒ *bool*
((- - - *sf*⊨ -) [*80*,*80*,*80*,*10*] *10*)
where

$$
(\sigma\ i\ j\ sf\models \mathit{sfrdiamond}\ f) = (\ j <\ \infty \wedge (\exists\, k.\ k = j \vee (k > j \wedge (\sigma\ j\ k\ f\models f))))
$$

fun *semantics-past-nl-itl* :: *interval* ⇒ *eint* ⇒ *eint* ⇒ *past-nl-itl* ⇒ *bool*

((- - - $p\models$ -) [80,80,80,10] 10)
where
(σ i j $p\models$ *pintro* f) = (σ i j $c\models$ f)
| (σ i j $p\models$ *pinot* f) = ($\neg$ (σ i j $p\models$ f))
| (σ i j $p\models$ (f *pior* g)) = ((σ i j $p\models$ f) $\vee$ (σ i j $p\models$ g))
| (σ i j $p\models$ *pldiamond* f) = ($i > -\infty \wedge (\exists k.\ k \leq i \wedge (\sigma\ k\ i\ p\models f))$)

fun *semantics-stricly-past-nl-itl* :: *interval* $\Rightarrow$ *eint* $\Rightarrow$ *eint* $\Rightarrow$ *strictly-past-nl-itl* $\Rightarrow$ *bool*
((- - - $sp\models$ -) [80,80,80,10] 10)
where
(σ i j $sp\models$ *spldiamond* f) = ($i > -\infty \wedge (\exists k.\ k = i \vee (k < i \wedge (\sigma\ k\ i\ p\models f))))$

fun *semantics-separated-nl-itl* :: *interval* $\Rightarrow$ *eint* $\Rightarrow$ *eint* $\Rightarrow$ *separated-nl-itl* $\Rightarrow$ *bool*
((- - - $s\models$ -) [80,80,80,10] 10)
where
(σ i j $s\models$ *past* f) = (σ i j $p\models$ f)
| (σ i j $s\models$ *future* f) = (σ i j $f\models$ f)
| (σ i j $s\models$ *sinot* f) = ($\neg$ (σ i j $s\models$ f))
| (σ i j $s\models$ (f *sior* g)) = ((σ i j $s\models$ f) $\vee$ (σ i j $s\models$ g))

lemma *similar-defs* :
similar $\sigma 0$ n $\sigma 1$ $\longleftrightarrow$ ($\forall$ k p . ($p \in (\sigma 0\ k) \wedge p \neq n$) $\longleftrightarrow$ ($p \in (\sigma 1\ k) \wedge p \neq n$))
unfolding *similar-def* **by** *auto*

lemma *csimilar-defs* [*simp*]:
csimilar $\sigma 0$ n i j $\sigma 1$ $\longleftrightarrow$
($\forall$ k p. *if* $i \leq k \wedge k \leq j$ *then* ($p \in (\sigma 0\ k) \wedge p \neq n$) $\longleftrightarrow$ ($p \in (\sigma 1\ k) \wedge p \neq n$)
else ($p \in (\sigma 0\ k)$) $\longleftrightarrow$ ($p \in (\sigma 1\ k)$))
unfolding *csimilar-def* **by** *auto*

lemma *similar-defs-alt*:
($\forall$ k . ($\sigma 0\ k$) $-$ $\{n\}$ = ($\sigma 1\ k$) $-$ $\{n\}$) $\longleftrightarrow$ *similar* $\sigma 0$ n $\sigma 1$
unfolding *similar-def* **by** *auto*

lemma *similars-defs* [*simp*]:
similars $\sigma 0$ A $\sigma 1$ $\longleftrightarrow$ ($\forall$ k p. $p \notin A \longrightarrow$ ($p \in (\sigma 0\ k) \longleftrightarrow p \in (\sigma 1\ k)$))
unfolding *similars-def* **by** *auto*

lemma *similars-defs-alt*:
similars $\sigma 0$ A $\sigma 1$ $\longleftrightarrow$ ($\forall$ k. ($\sigma 0\ k$) $-$ A = ($\sigma 1\ k$) $-$ A)
unfolding *similars-def* **by** *auto*

lemma *similar-similars*:
similar $\sigma 0$ n $\sigma 1$ = *similars* $\sigma 0$ $\{n\}$ $\sigma 1$
using *similar-defs-alt similars-defs-alt* **by** *presburger*

lemma *similars-mono*:
assumes $A \subseteq B$
similars $\sigma 0\ A\ \sigma 1$
shows *similars* $\sigma 0\ B\ \sigma 1$
using *assms* **by** *auto*

lemma *similar-refl*:
similar $\sigma\ n\ \sigma$
using *similar-defs-alt* **by** *blast*

lemma *csimilar-refl*:
csimilar $\sigma\ n\ i\ j\ \sigma$
by *simp*

lemma *similar-commute*:
similar $\sigma\ n\ \sigma' = similar\ \sigma'\ n\ \sigma$
by (*metis similar-defs-alt*)

lemma *csimilar-commute*:
csimilar $\sigma\ n\ i\ j\ \sigma' = csimilar\ \sigma'\ n\ i\ j\ \sigma$
by *simp blast*

lemma *similar-trans*:
assumes *similar* $\sigma\ n\ \sigma'$
similar $\sigma'\ n\ \sigma''$
shows *similar* $\sigma\ n\ \sigma''$
using *assms*
by (*metis similar-defs-alt*)

lemma *csimilar-trans*:
assumes *csimilar* $\sigma\ n\ i\ j\ \sigma'$
csimilar $\sigma'\ n\ i\ j\ \sigma''$
shows *csimilar* $\sigma\ n\ i\ j\ \sigma''$
using *assms* **by** *simp*

lemma *similars-refl*:
similars $\sigma\ A\ \sigma$
by *simp*

lemma *similars-commute*:
similars $\sigma\ A\ \sigma' = similars\ \sigma'\ A\ \sigma$
by *simp blast*

lemma *similars-trans*:
assumes *similars* $\sigma\ A\ \sigma'$

similars σ' *A* σ''
shows *similars* σ *A* σ''
using *assms* **by** *simp*

lemma *upd-add-sigma-defs* [*simp*]:
upd-add-sigma σ *k n* = $\sigma(k :=$ *insert n* $(\sigma\ k))$
unfolding *upd-add-sigma-def* **by** *simp*

lemma *upd-remove-sigma-defs* [*simp*]:
upd-remove-sigma σ *k A* = $\sigma(k :=\ (\sigma\ k) - A)$
unfolding *upd-remove-sigma-def* **by** *simp*

lemma *upd-remove-all-sigma-defs* [*simp*]:
upd-remove-all-sigma σ *A* = $(\lambda i.$ (*upd-remove-sigma* σ *i A*) *i*)
unfolding *upd-remove-all-sigma-def* **by** *simp*

lemma *similars-upd-remove-all*:
similars σ *A* (*upd-remove-all-sigma* σ *A*)
by *simp*

lemma *finite-bvars*:
finite (*bvars f*)
by (*induct f*) *simp-all*

lemma *finite-fvars*:
finite (*fvars f*)
by (*induct f*) *simp-all*

lemma *similar-upd*:
shows *similar* σ *n* (*upd* σ *n l*)
using *similar-defs-alt*[*of* σ *n* (*upd* σ *n l*)]
using *upd-def* **by** *force*

lemma *exist-sigma-exist-value*:
$(\exists\ \sigma'.\ (\sigma'\ i\ j \models f) \wedge (similar\ \sigma\ n\ \sigma')\ ) \longleftrightarrow (\exists\ l.\ (upd\ \sigma\ n\ l)\ i\ j \models f\ )$
proof −
have *2*: $(\exists\ \sigma'.\ (\sigma'\ i\ j \models f) \wedge (similar\ \sigma\ n\ \sigma')\ ) \Longrightarrow$
$(\exists\ l\ .\ ((upd\ \sigma\ n\ l)\ i\ j \models f))$
proof −
assume *a0*: $(\exists\ \sigma'.\ (\sigma'\ i\ j \models f) \wedge (similar\ \sigma\ n\ \sigma')\ )$

```
    show (∃ l . ((upd σ n l) i j ⊨ f))
     proof −
      obtain σ′ where 4: (σ′ i j ⊨ f) ∧ (similar σ n σ′)
        using a0 by blast
      have 6: σ′ = (upd σ′ n (λ k. n ∈ (σ′ k) ))
        unfolding upd-def
        by (metis Diff-empty Diff-insert0 insert-absorb)
      have 7: σ = (upd σ n (λ k. n ∈ (σ k) ))
         unfolding upd-def
        by (metis Diff-empty Diff-insert0 insert-absorb)
      have 8: ⋀x . (σ′ x) = (upd σ n (λ k. n ∈ (σ′ k) )) x
       unfolding upd-def using 4
       by (metis 6 insert-Diff-single similar-defs-alt upd-def)
      have 9: σ′ = (upd σ n (λ k. n ∈ (σ′ k) ))
         using 8 by blast
      have 10: (upd σ n (λ k. n ∈ (σ′ k) )) i j ⊨ f
        using 4 9 by auto
       show ?thesis using 10 by blast
     qed
    qed
  have 11: (∃ l . ((upd σ n l) i j ⊨ f)) ⟹
           (∃ σ′. (σ′ i j ⊨ f) ∧ (similar σ n σ′) )
  proof −
   assume a1: (∃ l . ((upd σ n l) i j ⊨ f))
   show (∃ σ′. (σ′ i j ⊨ f) ∧ (similar σ n σ′) )
   proof −
    obtain l where 12: ((upd σ n l) i j ⊨ f)
     using a1 by blast
    have 13:  similar σ n (upd σ n l)
     by (simp add: similar-upd)
    show ?thesis
    using 12 13 by blast
   qed
  qed
 show ?thesis
 using  11 2 by blast
qed


lemma cexist-sigma-cexist-value:
 (∃ σ′. (σ′ i j c⊨ f) ∧ (similar σ n σ′) ) ⟷ (∃ l. (upd σ n l) i j c⊨ f )
proof −
 have 2: (∃ σ′. (σ′ i j c⊨ f) ∧ (similar σ n σ′) ) ⟹
         (∃ l . ((upd σ n l) i j c⊨ f))
   proof −
    assume a0: (∃ σ′. (σ′ i j c⊨ f) ∧ (similar σ n σ′) )
    show (∃ l . ((upd σ n l) i j c⊨ f))
     proof −
      obtain σ′ where 4: (σ′ i j c⊨ f) ∧ (similar σ n σ′)
        using a0 by blast
```

```
      have 6: σ' = (upd σ' n (λ k. n ∈ (σ' k) ))
        unfolding upd-def
        by (metis Diff-empty Diff-insert0 insert-absorb)
      have 7: σ = (upd σ n (λ k. n ∈ (σ k) ))
         unfolding upd-def
        by (metis Diff-empty Diff-insert0 insert-absorb)
      have 8: ⋀x . (σ' x) = (upd σ n (λ k. n ∈ (σ' k) )) x
       unfolding upd-def using 4
       by (metis 6 insert-Diff-single similar-defs-alt upd-def)
      have 9: σ' = (upd σ n (λ k. n ∈ (σ' k) ))
         using 8 by blast
      have 10: (upd σ n (λ k. n ∈ (σ' k) )) i j c⊨ f
        using 4 9 by auto
       show ?thesis using 10 by blast
     qed
    qed
  have 11: (∃ l . ((upd σ n l) i j c⊨ f)) ⟹
           (∃ σ'. (σ' i j c⊨ f) ∧ (similar σ n σ') )
  proof −
   assume a1: (∃ l . ((upd σ n l) i j c⊨ f))
   show (∃ σ'. (σ' i j c⊨ f) ∧ (similar σ n σ') )
   proof −
    obtain l where 12: ((upd σ n l) i j c⊨ f)
     using a1 by blast
    have 13:  similar σ n (upd σ n l)
    by (simp add: similar-upd)
    show ?thesis
    using 12 13 by blast
   qed
  qed
 show ?thesis
 using  11 2 by blast
qed

lemma upd-absorb:
 assumes  x1 = x2
 shows  upd (upd σ x1 l1) x2 l2 = (upd σ x2 l2)
proof −
 have 1: ⋀ i. (upd (upd σ x1 l1) x2 l2 ) i = (upd σ x2 l2) i
  using assms unfolding upd-def by auto
 show ?thesis using 1 by auto
qed

lemma upd-swap:
 assumes  x1 ≠ x2
 shows  upd (upd σ x1 l1) x2 l2 = (upd (upd σ x2 l2) x1 l1)
proof −
 have 1: ⋀ i. (upd (upd σ x1 l1) x2 l2) i = (upd (upd σ x2 l2) x1 l1) i
   using assms unfolding upd-def by auto
 show ?thesis using 1 by auto
```

qed

lemma *not-fvar-upd*:
assumes $n \notin fvars\ f$
shows $(\sigma\ i\ j\ \models f) = ((upd\ \sigma\ n\ l)\ i\ j \models f)$
using *assms*
proof (*induction f arbitrary*: $\sigma\ l\ i\ j$)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case*
by (*metis fvars.simps*(*2*) *insertCI semantics-nl-itl.simps*(*2*) *similar-defs similar-upd*)
next
case (*iimp-d f1 f2*)
then show *?case* **by** *simp*
next
case (*next-d f*)
then show *?case* **by** *auto*
next
case (*chop-d f1 f2*)
then show *?case* **by** *simp*
next
case (*ldiamond-d f*)
then show *?case* **by** *simp*
next
case (*rdiamond-d f*)
then show *?case* **by** *simp*
next
case (*chopstar-d f*)
then show *?case*
by *force*
next
case (*exists-d x1 f*)
then show *?case*
proof −
have *e1*: $(\sigma\ i\ j \models Ex\ x1.\ f) = (\exists\ l.\ ((upd\ \sigma\ x1\ l)\ i\ j \models f)\ )$
using *exist-sigma-exist-value* **by** *auto*
have *e2*: $((upd\ \sigma\ n\ l)\ i\ j \models Ex\ x1.\ f) =$
$(\exists\ l1.\ ((upd\ (upd\ \sigma\ n\ l)\ x1\ l1)\ i\ j \models f)\ )$
using *exist-sigma-exist-value* **by** *force*
have *e3*: $(\exists\ l.\ ((upd\ \sigma\ x1\ l)\ \ i\ j \models f)\ ) \Longrightarrow$
$(\exists\ l1.\ ((upd\ (upd\ \sigma\ n\ l)\ x1\ l1)\ \ i\ j \models f)\ )$
by (*metis Diff-iff exists-d.IH exists-d.prems fvars.simps*(*9*) *singletonD upd-absorb upd-swap*)
have *e4*: $(\exists\ l1.\ ((upd\ (upd\ \sigma\ n\ l)\ x1\ l1)\ \ i\ j \models f)\ ) \Longrightarrow$
$(\exists\ l.\ ((upd\ \sigma\ x1\ l)\ i\ j\ \models f)\ )$
by (*metis Diff-iff emptyE exists-d.IH exists-d.prems fvars.simps*(*9*) *insertE upd-absorb upd-swap*)
show *?thesis*
using *e1 e2 e3 e4* **by** *blast*

qed
qed

lemma *not-fvar-upd-introspective*:
assumes $n \notin cfvars\ f$
shows $(\sigma\ i\ j\ \ c\models f) = ((upd\ \sigma\ n\ l)\ i\ j\ c\models f)$
using *assms*
proof (*induction f arbitrary*: $\sigma\ l\ i\ j$)
case *cfalse-d*
then show *?case* **by** *simp*
next
case (*catom-d x*)
then show *?case*
by (*metis cfvars.simps*(*2*) *insertCI semantics-introspective-nl-itl.simps*(*2*) *similar-defs similar-upd*)
next
case (*ciimp-d f1 f2*)
then show *?case* **by** *simp*
next
case (*cnext-d f*)
then show *?case* **by** *auto*
next
case (*cchop-d f1 f2*)
then show *?case* **by** *simp*
next
case (*cchopstar-d f*)
then show *?case* **by** *force*
next
case (*cexists-d x1 f*)
then show *?case*
proof −
have *e1*: $(\sigma\ i\ j\ c\models cEx\ x1.\ f) = (\exists\ l.\ ((upd\ \sigma\ x1\ l)\ i\ j\ c\models f)\ )$
using *cexist-sigma-cexist-value* **by** *auto*
have *e2*: $((upd\ \sigma\ n\ l)\ i\ j\ c\models cEx\ x1.\ f) =$
$(\exists\ l1.\ \ ((upd\ (upd\ \sigma\ n\ l)\ x1\ l1)\ i\ j\ c\models f)\ )$
using *cexist-sigma-cexist-value* **by** *force*
have *e3*: $(\exists\ l.\ \ ((upd\ \sigma\ x1\ l)\ \ i\ j\ c\models f)\ ) \Longrightarrow$
$(\exists\ l1.\ \ ((upd\ (upd\ \sigma\ n\ l)\ x1\ l1)\ \ i\ j\ c\models f)\ )$
by (*metis Diff-iff cexists-d.IH cexists-d.prems cfvars.simps*(*7*) *singletonD upd-absorb upd-swap*)
have *e4*: $(\exists\ l1.\ \ ((upd\ (upd\ \sigma\ n\ l)\ x1\ l1)\ \ i\ j\ c\models f)\ ) \Longrightarrow$
$(\exists\ l.\ \ ((upd\ \sigma\ x1\ l)\ i\ j\ \ c\models f)\ )$
by (*metis Diff-iff emptyE cexists-d.IH cexists-d.prems cfvars.simps*(*7*) *insertE upd-absorb upd-swap*)
show *?thesis*
using *e1 e2 e3 e4* **by** *blast*
qed
qed

lemma *inot-defs* [*simp*]:

$(\sigma\ i\ j \models inot\ f\ ) = (\neg(\sigma\ i\ j \models f))$
by (*simp add*: *inot-d-def*)

lemma *cinot-defs* [*simp*]:
$(\sigma\ i\ j\ c\models cinot\ f\ ) = (\neg(\sigma\ i\ j\ c\models f))$
by (*simp add*: *cinot-d-def*)

lemma *ior-defs* [*simp*]:
$(\sigma\ i\ j \models f1\ ior\ f2) = (\ (\sigma\ i\ j \models f1) \vee (\sigma\ i\ j \models f2))$
by (*metis inot-defs ior-d-def semantics-nl-itl.simps*(*3*))

lemma *cior-defs* [*simp*]:
$(\sigma\ i\ j\ c\models f1\ cior\ f2) = (\ (\sigma\ i\ j\ c\models f1) \vee (\sigma\ i\ j\ c\models f2))$
by (*metis cinot-defs cior-d-def semantics-introspective-nl-itl.simps*(*3*))

lemma *iand-defs* [*simp*]:
$(\sigma\ i\ j \models f1\ iand\ f2) = (\ (\sigma\ i\ j \models f1) \wedge (\sigma\ i\ j \models f2))$
by (*simp add*: *iand-d-def*)

lemma *ciand-defs* [*simp*]:
$(\sigma\ i\ j\ c\models f1\ ciand\ f2) = (\ (\sigma\ i\ j\ c\models f1) \wedge (\sigma\ i\ j\ c\models f2))$
by (*simp add*: *ciand-d-def*)

lemma *ieqv-defs* [*simp*]:
$(\sigma\ i\ j \models f1\ ieqv\ f2) = (\ (\sigma\ i\ j \models f1) = (\sigma\ i\ j \models f2))$
by (*metis iand-defs ieqv-d-def semantics-nl-itl.simps*(*3*))

lemma *cieqv-defs* [*simp*]:
$(\sigma\ i\ j\ c\models f1\ cieqv\ f2) = (\ (\sigma\ i\ j\ c\models f1) = (\sigma\ i\ j\ c\models f2))$
by (*metis ciand-defs cieqv-d-def semantics-introspective-nl-itl.simps*(*3*))

lemma *itrue-defs* [*simp*]:
$(\sigma\ i\ j \models itrue)$
by (*simp add*: *itrue-d-def*)

lemma *citrue-defs* [*simp*]:
$(\sigma\ i\ j\ c\models citrue)$
by (*simp add*: *citrue-d-def*)

lemma *empty-defs* [*simp*]:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models empty) = (\ i = j\ )$
using *assms*
by (*auto simp add*: *empty-d-def more-d-def*)
(*metis eint-bot eint-cases eint-less-eq*(*2*) *nle-le*)

lemma *cempty-defs* [*simp*]:
assumes $i \leq j$

$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models cempty) = (\ i = j\ )$
using *assms*
by (*auto simp add*: *cempty-d-def cmore-d-def*)
(*metis eint-bot eint-cases eint-less-eq*(*2*) *nle-le*)

lemma *skip-defs* [*simp*]:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models skip) = (\ i{+}(eint\ 1) = j)$
proof −
have *1*: $(\sigma\ i\ j \models skip) = (\sigma\ i\ j\ \models next\ empty)$
unfolding *skip-d-def* **by** *simp*
have *2*: $(\sigma\ i\ j\ \models next\ empty) = (\ i \neq -\infty \wedge i{<}j \wedge (\ \sigma\ (i + (eint\ 1)\ )\ j \models empty))$
by *simp*
have *3*: $i \neq -\infty \wedge i{<}j \longrightarrow (i + (eint\ 1)) \leq j$
by (*metis abs-eint.cases assms*(*2*) *eint-less-le-alt plus-eint.simps*(*1*))
have *4*: $i + eint\ (1{::}int) \neq \infty$
by (*simp add*: *assms*(*2*))
have *5*: $(\ i \neq -\infty \wedge i{<}j \wedge (\ \sigma\ (i{+}\ (eint\ 1)\ )\ j \models empty)) \longrightarrow ((i + (eint\ 1)\ ) = j)$
using *assms empty-defs*[*of* $i{+}(eint\ 1)\ j$] *3 4* **by** *blast*
have *6*: $(\ i{+}(eint\ 1) = j) \longrightarrow (\ i \neq -\infty \wedge i{<}j \wedge (\ \sigma\ (i + (eint\ 1)\ )\ j \models empty))$
using *assms empty-defs*[*of* $i{+}(eint\ 1)\ j$]
by (*metis 4 eint-0-less-1 eint-between*(*2*) *eint-infinity-cases*
less-eq-eint-def one-eint-def plus-eint.simps(*5*))
show *?thesis* **using** *1 2 5 6* **by** *blast*
qed

lemma *cskip-defs* [*simp*]:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models cskip) = (\ i{+}(eint\ 1) = j)$
proof −
have *1*: $(\sigma\ i\ j\ c\models cskip) = (\sigma\ i\ j\ \ c\models cnext\ cempty)$
unfolding *cskip-d-def* **by** *simp*
have *2*: $(\sigma\ i\ j\ \ c\models cnext\ cempty) = (\ i \neq -\infty \wedge i{<}j \wedge (\ \sigma\ (i + (eint\ 1)\ )\ j\ c\models cempty))$
by *simp*
have *3*: $i \neq -\infty \wedge i{<}j \longrightarrow (i + (eint\ 1)) \leq j$
by (*metis abs-eint.cases assms*(*2*) *eint-less-le-alt plus-eint.simps*(*1*))
have *4*: $i + eint\ (1{::}int) \neq \infty$
by (*simp add*: *assms*(*2*))
have *5*: $(\ i \neq -\infty \wedge i{<}j \wedge (\ \sigma\ (i{+}\ (eint\ 1)\ )\ j\ c\models cempty)) \longrightarrow ((i + (eint\ 1)\ ) = j)$
using *assms cempty-defs*[*of* $i{+}(eint\ 1)\ j$] *3 4* **by** *blast*
have *6*: $(\ i{+}(eint\ 1) = j) \longrightarrow (\ i \neq -\infty \wedge i{<}j \wedge (\ \sigma\ (i + (eint\ 1)\ )\ j\ c\models cempty))$
using *assms cempty-defs*[*of* $i{+}(eint\ 1)\ j$]
by (*metis 4 eint-0-less-1 eint-between*(*2*) *eint-infinity-cases*
less-eq-eint-def one-eint-def plus-eint.simps(*5*))

show *?thesis* **using** *1 2 5 6* **by** *blast*
qed

lemma *lfinite-defs*[*simp*]:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{lfinite}) = (i \neq -\infty)$
using *assms*
by (*auto simp add*: *empty-d-def more-d-def lfinite-d-def*)
(*metis eint-bot nle-le*)

lemma *clfinite-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models \mathit{clfinite}) = (i \neq -\infty)$
using *assms*
by (*auto simp add*: *cempty-d-def cmore-d-def clfinite-d-def*)
(*metis eint-bot nle-le*)

lemma *rfinite-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{rfinite}) = (j \neq \infty)$
using *assms*
by (*auto simp add*: *empty-d-def more-d-def rfinite-d-def*)
(*metis eint-cases eint-less-eq*(*3*) *nle-le*)

lemma *crfinite-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models \mathit{crfinite}) = (j \neq \infty)$
using *assms*
by (*auto simp add*: *cempty-d-def cmore-d-def crfinite-d-def*)
(*metis eint-cases eint-less-eq*(*3*) *nle-le*)

lemma *prev-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{prev}\ f) = (j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (\mathit{eint}\ 1)) \models f))$
proof −
have *1*: $(\sigma\ i\ j \models \mathit{prev}\ f) =$
$(\exists k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ i\ (\mathit{eint}\ k) \models f) \wedge (\sigma\ (\mathit{eint}\ k)\ j \models \mathit{skip}))$
unfolding *prev-d-def* **by** *simp*
have *2*: $(\exists k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ i\ (\mathit{eint}\ k) \models f) \wedge (\sigma\ (\mathit{eint}\ k)\ j \models \mathit{skip})) \longrightarrow$

$(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1)) \models f))$
using *assms(3) le-eint-less* **by** *auto*
have *3*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1)) \models f)) \longrightarrow (\exists k.\ j = (eint\ k))$
by (*metis less-eint.elims(2)*)
have *4*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1)) \models f)) \longrightarrow$
$i \leq eint\ (int\text{-}of\text{-}eint\ (j-1))$
using *assms 3* **using** *int-less-eint-iff int-of-eint-minus* **by** *fastforce*
have *5*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1)) \models f)) \longrightarrow$
$eint\ (int\text{-}of\text{-}eint\ (j-1)) \leq j \wedge$
$(\sigma\ i\ (eint\ (int\text{-}of\text{-}eint\ (j-1))) \models f) \wedge (\sigma\ (eint\ (int\text{-}of\text{-}eint\ (j-1)))\ j \models skip)$

using *assms 3* **by** (*auto simp add*: *one-eint-def*)
have *6*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1)) \models f)) \longrightarrow$
$(\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k) \models f) \wedge (\sigma\ (eint\ k)\ j \models skip))$
using *4 5* **by** *fastforce*
show *?thesis* **using** *2 6 1* **by** *blast*
qed

lemma *cprev-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models cprev\ f) = (j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f))$
proof −
have *1*: $(\sigma\ i\ j\ c\models cprev\ f) =$
$(\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k)\ c\models f) \wedge (\sigma\ (eint\ k)\ j\ c\models cskip))$
unfolding *cprev-d-def* **by** *simp*
have *2*: $(\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k)\ c\models f) \wedge (\sigma\ (eint\ k)\ j\ c\models cskip)) \longrightarrow$
$(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f))$
using *assms(3) le-eint-less* **by** *auto*
have *3*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f)) \longrightarrow (\exists k.\ j = (eint\ k))$
by (*metis less-eint.elims(2)*)
have *4*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f)) \longrightarrow$
$i \leq eint\ (int\text{-}of\text{-}eint\ (j-1))$
using *assms 3* **using** *int-less-eint-iff int-of-eint-minus* **by** *fastforce*
have *5*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f)) \longrightarrow$
$eint\ (int\text{-}of\text{-}eint\ (j-1)) \leq j \wedge$
$(\sigma\ i\ (eint\ (int\text{-}of\text{-}eint\ (j-1)))\ c\models f) \wedge (\sigma\ (eint\ (int\text{-}of\text{-}eint\ (j-1)))\ j\ c\models cskip)$

using *assms 3* **by** (*auto simp add*: *one-eint-def*)
have *6*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f)) \longrightarrow$
$(\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k)\ c\models f) \wedge (\sigma\ (eint\ k)\ j\ c\models cskip))$
using *4 5* **by** *fastforce*
show *?thesis* **using** *2 6 1* **by** *blast*
qed

lemma *wnext-defs* :
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$

shows $(\sigma\ i\ j \models wnext\ f) = (\ i = -\infty \vee i = j \vee (i < j \wedge (\sigma\ (i + eint\ 1)\ j \models f)))$
proof −
have *2*: $(\sigma\ i\ j \models inot\ (next\ inot\ f)) = (\neg(\sigma\ i\ j \models\ (next\ inot\ f)))$
using *inot-defs* **by** *blast*
have *3*: $(\sigma\ i\ j \models\ (next\ inot\ f)) = (i \neq -\infty \wedge i < j \wedge \neg\ (\sigma\ (i + eint\ 1)\ j \models f))$
by *simp*
have *4*: $(\neg(\sigma\ i\ j \models\ (next\ inot\ f))) =$
$(\neg(i \neq -\infty \wedge i < j \wedge \neg\ (\sigma\ (i + eint\ 1)\ j \models f)))$
using *3* **by** *blast*
have *5*: $(\neg(i \neq -\infty \wedge i < j \wedge \neg\ (\sigma\ (i + eint\ 1)\ j \models f))) =$
$(\ i = -\infty \vee i = j \vee (i < j \wedge (\sigma\ (i + eint\ 1)\ j \models f)))$
using *assms(1)* **by** *auto*
show *?thesis* **using** *2 3 5 wnext-d-def* **by** *presburger*
qed

lemma *cwnext-defs* :
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models cwnext\ f) = (\ i = -\infty \vee i = j \vee (i < j \wedge (\sigma\ (i + eint\ 1)\ j\ c\models f)))$
proof −
have *2*: $(\sigma\ i\ j\ c\models cinot\ (cnext\ cinot\ f)) = (\neg(\sigma\ i\ j\ c\models\ (cnext\ cinot\ f)))$
using *cinot-defs* **by** *blast*
have *3*: $(\sigma\ i\ j\ c\models\ (cnext\ cinot\ f)) = (i \neq -\infty \wedge i < j \wedge \neg\ (\sigma\ (i + eint\ 1)\ j\ c\models f))$
by *simp*
have *4*: $(\neg(\sigma\ i\ j\ c\models\ (cnext\ cinot\ f))) =$
$(\neg(i \neq -\infty \wedge i < j \wedge \neg\ (\sigma\ (i + eint\ 1)\ j\ c\models f)))$
using *3* **by** *blast*
have *5*: $(\neg(i \neq -\infty \wedge i < j \wedge \neg\ (\sigma\ (i + eint\ 1)\ j\ c\models f))) =$
$(\ i = -\infty \vee i = j \vee (i < j \wedge (\sigma\ (i + eint\ 1)\ j\ c\models f)))$
using *assms(1)* **by** *auto*
show *?thesis* **using** *2 3 5 cwnext-d-def* **by** *presburger*
qed

lemma *wprev-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models wprev\ f) = (j = \infty \vee i = j \vee (i < j \wedge (\sigma\ i\ (j - (eint\ 1)) \models f)))$
proof −
have *1*: $(\sigma\ i\ j \models wprev\ f) = (\sigma\ i\ j \models inot\ (prev\ inot\ f))$
unfolding *wprev-d-def* **by** *blast*
have *2*: $(\sigma\ i\ j \models inot\ (prev\ inot\ f)) = (\neg\ (\sigma\ i\ j \models prev\ inot\ f))$
using *assms inot-defs[of σ i j prev inot f]* **by** *blast*
have *3*: $(\sigma\ i\ j \models prev\ inot\ f) = (j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - eint\ 1) \models inot\ f))$
using *assms prev-defs[of i j σ inot f]* **by** *blast*
have *4*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - eint\ 1) \models inot\ f)) =$
$(j \neq \infty \wedge i < j \wedge \neg(\sigma\ i\ (j - eint\ 1) \models\ f))$
by *simp*
have *5*: $(\neg\ (\sigma\ i\ j \models prev\ inot\ f)) =$

$(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma\ i\ (j - eint\ 1) \models f)))$
using *3 4* **by** *presburger*
have *6*: $(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma\ i\ (j - eint\ 1) \models f))) =$
$(j = \infty \vee i = j \vee (i < j \wedge (\sigma\ i\ (j - (eint\ 1)) \models f)))$
using *assms(1) order-less-le* **by** *blast*
from *1 2 5 6* **show** *?thesis*
by *presburger*
qed

lemma *cwprev-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models cwprev\ f) = (j = \infty \vee i = j \vee (i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f)))$
proof −
have *1*: $(\sigma\ i\ j\ c\models cwprev\ f) = (\sigma\ i\ j\ c\models cinot\ (cprev\ cinot\ f))$
unfolding *cwprev-d-def* **by** *blast*
have *2*: $(\sigma\ i\ j\ c\models cinot\ (cprev\ cinot\ f)) = (\neg\ (\sigma\ i\ j\ c\models cprev\ cinot\ f))$
using *assms cinot-defs*[*of* σ *i j cprev cinot f*] **by** *blast*
have *3*: $(\sigma\ i\ j\ c\models cprev\ cinot\ f) = (j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - eint\ 1)\ c\models cinot\ f))$
using *assms cprev-defs*[*of i j* σ *cinot f*] **by** *blast*
have *4*: $(j \neq \infty \wedge i < j \wedge (\sigma\ i\ (j - eint\ 1)\ c\models cinot\ f)) =$
$(j \neq \infty \wedge i < j \wedge \neg(\sigma\ i\ (j - eint\ 1)\ c\models f))$
by *simp*
have *5*: $(\neg\ (\sigma\ i\ j\ c\models cprev\ cinot\ f)) =$
$(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma\ i\ (j - eint\ 1)\ c\models f)))$
using *3 4* **by** *presburger*
have *6*: $(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma\ i\ (j - eint\ 1)\ c\models f))) =$
$(j = \infty \vee i = j \vee (i < j \wedge (\sigma\ i\ (j - (eint\ 1))\ c\models f)))$
using *assms(1) order-less-le* **by** *blast*
from *1 2 5 6* **show** *?thesis*
by *presburger*
qed

lemma *not-atom-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models inot\ (\$p)) = (i = -\infty \vee\ (p \notin(\sigma\ (int\text{-}of\text{-}eint\ i))))$
using *assms* **by** *simp*

lemma *not-catom-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models cinot\ (\$\$p)) = (i = -\infty \vee\ (p \notin(\sigma\ (int\text{-}of\text{-}eint\ i))))$
using *assms* **by** *simp*

lemma *diamond-defs*:

assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{diamond}\ f) = (\exists k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ (\mathit{eint}\ k)\ j \models f))$
using *assms* **unfolding** *sometimes-d-def* **by** *simp*

lemma *cdiamond-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models \mathit{cdiamond}\ f) = (\exists k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ (\mathit{eint}\ k)\ j\ c\models f))$
using *assms* **unfolding** *csometimes-d-def* **by** *simp*

lemma *box-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{box}\ f) = (\forall k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \longrightarrow (\sigma\ (\mathit{eint}\ k)\ j \models f))$
unfolding *always-d-def sometimes-d-def* **using** *assms* **by** *auto*

lemma *cbox-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models \mathit{cbox}\ f) = (\forall k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \longrightarrow (\sigma\ (\mathit{eint}\ k)\ j\ c\models f))$
unfolding *calways-d-def csometimes-d-def* **using** *assms* **by** *auto*

lemma *di-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{di}\ f) = (\exists k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ i\ (\mathit{eint}\ k) \models f))$
using *assms* **unfolding** *di-d-def* **by** *simp*

lemma *cdi-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models \mathit{cdi}\ f) = (\exists k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ i\ (\mathit{eint}\ k)\ c\models f))$
using *assms* **unfolding** *cdi-d-def* **by** *simp*

lemma *bi-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{bi}\ f) = (\forall k.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \longrightarrow (\sigma\ i\ (\mathit{eint}\ k)\ \models f))$
unfolding *bi-d-def di-d-def* **using** *assms* **by** *auto*

lemma *cbi-defs*:
assumes $i \leq j$

$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models\ cbi\ f) = (\forall\, k.\ i \leq eint\ k \wedge eint\ k \leq j \longrightarrow (\sigma\ i\ (eint\ k)\ \ c\models f)\ )$
unfolding *cbi-d-def cdi-d-def* **using** *assms* **by** *auto*

lemma *init-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models init\ f) = (i \neq -\infty \wedge (\sigma\ i\ i \models f))$
proof −
have *1*: $(\sigma\ i\ j \models init\ f) =$
$(\exists\, k{::}int.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k) \models empty) \wedge (\sigma\ i\ (eint\ k) \models f))$
unfolding *init-d-def* **by** *simp*
have *2*: $(\exists\, k{::}int.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k) \models empty) \wedge (\sigma\ i\ (eint\ k) \models f))$
$\longrightarrow (i \neq \infty \wedge (\sigma\ i\ i \models f))$
using *assms(2)* **by** *force*
have *3*: $i \neq -\infty \wedge (\sigma\ i\ i \models f) \longrightarrow (\exists\ k.\ i = (eint\ k))$
by (*meson assms(2) int-of-eint.elims*)
have *4*: $i \neq -\infty \wedge (\sigma\ i\ i \models f) \longrightarrow$
$i \leq eint\ (int\text{-}of\text{-}eint\ i) \wedge eint\ (int\text{-}of\text{-}eint\ i) \leq j \wedge$
$(\sigma\ i\ (eint\ (int\text{-}of\text{-}eint\ i)) \models empty) \wedge (\sigma\ i\ (eint\ (int\text{-}of\text{-}eint\ i)) \models f)$
using *assms 3* **by** *fastforce*
have *5*: $i \neq -\infty \wedge (\sigma\ i\ i \models f) \longrightarrow$
$(\exists\, k{::}int.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k) \models empty) \wedge (\sigma\ i\ (eint\ k) \models f))$
using *4* **by** *fastforce*
show *?thesis* **using** *1 2 4* **by** *auto*
qed

lemma *cinit-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models\ cinit\ f) = (i \neq -\infty \wedge (\sigma\ i\ i\ c\models f))$
proof −
have *1*: $(\sigma\ i\ j\ c\models\ cinit\ f) =$
$(\exists\, k{::}int.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k)\ c\models\ cempty) \wedge (\sigma\ i\ (eint\ k)\ c\models f))$
unfolding *cinit-d-def* **by** *simp*
have *2*: $(\exists\, k{::}int.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k)\ c\models\ cempty) \wedge (\sigma\ i\ (eint\ k)\ c\models f))$
$\longrightarrow (i \neq \infty \wedge (\sigma\ i\ i\ c\models f))$
using *assms(2)* **by** *force*
have *3*: $i \neq -\infty \wedge (\sigma\ i\ i\ c\models f) \longrightarrow (\exists\ k.\ i = (eint\ k))$
by (*meson assms(2) int-of-eint.elims*)
have *4*: $i \neq -\infty \wedge (\sigma\ i\ i\ c\models f) \longrightarrow$
$i \leq eint\ (int\text{-}of\text{-}eint\ i) \wedge eint\ (int\text{-}of\text{-}eint\ i) \leq j \wedge$
$(\sigma\ i\ (eint\ (int\text{-}of\text{-}eint\ i))\ c\models\ cempty) \wedge (\sigma\ i\ (eint\ (int\text{-}of\text{-}eint\ i))\ c\models f)$
using *assms 3* **by** *fastforce*
have *5*: $i \neq -\infty \wedge (\sigma\ i\ i\ c\models f) \longrightarrow$
$(\exists\, k{::}int.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k)\ c\models\ cempty) \wedge (\sigma\ i\ (eint\ k)\ c\models f))$
using *4* **by** *fastforce*

show *?thesis* **using** *1 2 4* **by** *auto*
qed

lemma *fin-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models \mathit{fin}\ f) = (j \neq \infty \wedge (\sigma\ j\ j \models f))$
proof −
have *1*: $(\sigma\ i\ j \models \mathit{fin}\ f) =$
$(\exists k{::}\mathit{int}.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ (\mathit{eint}\ k)\ j \models \mathit{empty}) \wedge (\sigma\ (\mathit{eint}\ k)\ j \models f))$
unfolding *fin-d-def* **by** *simp*
have *2*: $(\exists k{::}\mathit{int}.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ (\mathit{eint}\ k)\ j \models \mathit{empty}) \wedge (\sigma\ (\mathit{eint}\ k)\ j \models f)) \longrightarrow$
$(j \neq \infty \wedge (\sigma\ j\ j \models f))$
using *PInfty-neq-eint*(*1*) *assms*(*3*) *empty-defs* **by** *blast*
have *3*: $(j \neq \infty \wedge (\sigma\ j\ j \models f)) \longrightarrow (\exists k.\ j = \mathit{eint}\ k)$
using *assms* **by** (*metis MInfty-eq-minfinity PInfty-eq-infinity eint.exhaust*)
have *4*: $(j \neq \infty \wedge (\sigma\ j\ j \models f)) \longrightarrow$
$i \leq \mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j) \wedge \mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j) \leq j \wedge$
$(\sigma\ (\mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j))\ j \models \mathit{empty}) \wedge (\sigma\ (\mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j))\ j \models f)$
using *assms 3* **by** *fastforce*
show *?thesis* **using** *1 2 4* **by** *auto*
qed

lemma *cfin-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models \mathit{cfin}\ f) = (j \neq \infty \wedge (\sigma\ j\ j\ c\models f))$
proof −
have *1*: $(\sigma\ i\ j\ c\models \mathit{cfin}\ f) =$
$(\exists k{::}\mathit{int}.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ (\mathit{eint}\ k)\ j\ c\models \mathit{cempty}) \wedge (\sigma\ (\mathit{eint}\ k)\ j\ c\models f))$
unfolding *cfin-d-def* **by** *simp*
have *2*: $(\exists k{::}\mathit{int}.\ i \leq \mathit{eint}\ k \wedge \mathit{eint}\ k \leq j \wedge (\sigma\ (\mathit{eint}\ k)\ j\ c\models \mathit{cempty}) \wedge (\sigma\ (\mathit{eint}\ k)\ j\ c\models f)) \longrightarrow$
$(j \neq \infty \wedge (\sigma\ j\ j\ c\models f))$
using *PInfty-neq-eint*(*1*) *assms*(*3*) *cempty-defs* **by** *blast*
have *3*: $(j \neq \infty \wedge (\sigma\ j\ j\ c\models f)) \longrightarrow (\exists k.\ j = \mathit{eint}\ k)$
using *assms* **by** (*metis MInfty-eq-minfinity PInfty-eq-infinity eint.exhaust*)
have *4*: $(j \neq \infty \wedge (\sigma\ j\ j\ c\models f)) \longrightarrow$
$i \leq \mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j) \wedge \mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j) \leq j \wedge$
$(\sigma\ (\mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j))\ j\ c\models \mathit{cempty}) \wedge (\sigma\ (\mathit{eint}\ (\mathit{int\text{-}of\text{-}eint}\ j))\ j\ c\models f)$
using *assms 3* **by** *fastforce*
show *?thesis* **using** *1 2 4* **by** *auto*
qed

lemma *state-nl-itl-defs*:
assumes *state-nl-itl w*
$i \neq \infty$
$j \neq -\infty$

shows $(\sigma\ i\ j \models w) \longleftrightarrow (\sigma\ i\ i \models w)$
using *assms*
by (*induct w arbitrary*: σ) *simp-all*

lemma *next-diamond*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models next\ (diamond\ f)) = (\sigma\ i\ j \models (lfinite\ iand\ diamond\ (next\ f)))$
proof −
have *1*: $(\sigma\ i\ j \models next\ (diamond\ f)) = (i \neq -\infty \wedge i < j \wedge (\sigma\ (i + eint\ 1)\ j \models diamond\ f))$
by *simp*
have *2*: $(i \neq -\infty \wedge i < j \wedge (\sigma\ (i + eint\ 1)\ j \models diamond\ f)) =$
$(\ i \neq -\infty \wedge i < j \wedge (\exists k.\ i + eint\ 1 \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ (eint\ k)\ j \models f)))$
using *diamond-defs*[*of* $(i + eint\ 1)\ j\ \sigma\ f$]
using *assms*(*3*) *less-eint.elims*(*2*) **by** *fastforce*
have *30*: $(\ i \neq -\infty \wedge i < j \wedge (\exists k.\ i + eint\ 1 \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ (eint\ k)\ j \models f))) \Longrightarrow$
$(i{\neq}{-}\infty \wedge\ (\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge eint\ k < j \wedge (\sigma\ (eint\ (k + 1))\ j \models f)))$
proof −
assume *a0*: $(\ i \neq -\infty \wedge i < j \wedge (\exists k.\ i + eint\ 1 \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ (eint\ k)\ j \models f)))$
show $(\ i{\neq}{-}\infty \wedge (\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge eint\ k < j \wedge (\sigma\ (eint\ (k + 1))\ j \models f)))$
proof −
obtain *k* **where** *31*: $i + eint\ 1 \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ (eint\ k)\ j \models f)$
using *a0* **by** *blast*
have *32*: $i \leq eint\ (k{-}1)$
using *31 assms*(*2*) *less-eint.elims*(*2*) **by** *fastforce*
have *33*: $eint\ (k{-}1) \leq j \wedge eint\ (k{-}1) < j$
by (*metis 31 diff-add-cancel eint-le-less less-add-one order-less-imp-le*)
have *34*: $(\sigma\ (eint\ ((k{-}1) + 1))\ j \models f)$
by (*simp add*: *31*)
show *?thesis*
using *32 33 34 a0* **by** *blast*
qed
qed
have *35*: $(i{\neq}{-}\infty \wedge\ (\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge eint\ k < j \wedge (\sigma\ (eint\ (k + 1))\ j \models f))) \Longrightarrow$
$(\ i \neq -\infty \wedge i < j \wedge (\exists k.\ i + eint\ 1 \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ (eint\ k)\ j \models f)))$
proof −
assume *a1*: $(i{\neq}{-}\infty \wedge\ (\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge eint\ k < j \wedge (\sigma\ (eint\ (k + 1))\ j \models f)))$
show $(\ i \neq -\infty \wedge i < j \wedge (\exists k.\ i + eint\ 1 \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ (eint\ k)\ j \models f)))$
proof −
obtain *k* **where** *36*: $i \leq eint\ k \wedge eint\ k \leq j \wedge eint\ k < j \wedge (\sigma\ (eint\ (k + 1))\ j \models f)$
using *a1* **by** *blast*
have *37*: $i + eint\ 1 \leq eint\ (k{+}1)$
by (*metis 36 add.commute add-mono eint-eq-1*(*1*) *eint-plus-1*(*1*) *order-refl*)
have *38*: $eint\ (k{+}1) \leq j$
by (*simp add*: *36 eint-less-le-alt*)
have *39*: $(\sigma\ (eint\ (k{+}1))\ j \models f)$
by (*simp add*: *36*)

```
      have 40: (∃ k. i + eint 1 ≤ eint k ∧ eint k ≤ j ∧ (σ (eint k) j ⊨ f))
        using 36 37 38 by auto
      have 41: i < j
        using a1 by fastforce
      have 42: i ≠ −∞
        by (simp add: a1)
     show ?thesis
     by (simp add: 40 41 42)
    qed
  qed
 have 4: (i≠−∞ ∧ (∃ k. i ≤ eint k ∧ eint k ≤ j ∧ (σ (eint k) j ⊨ next f))) =
       (i≠−∞ ∧ (∃ k. i ≤ eint k ∧ eint k ≤ j ∧ eint k < j ∧ (σ (eint (k + 1)) j ⊨ f)))
   by simp
 have 5: (σ i j ⊨ (lfinite iand diamond (next f))) =
         (i≠−∞ ∧(∃ k. i ≤ eint k ∧ eint k ≤ j ∧ (σ (eint k) j ⊨ next f)))
      using diamond-defs[of i j σ next f]  lfinite-defs assms by fastforce
 show ?thesis
 by (metis 1 2 30 35 4 5)
qed



lemma linf-defs[simp]:
 assumes  i ≠ ∞
       j ≠ −∞
       i ≤ j
shows  (σ i j ⊨ linf) = (i = −∞)
using assms unfolding linf-d-def by (simp)

lemma rinf-defs[simp]:
assumes  i ≠ ∞
       j ≠ −∞
       i ≤ j
shows  (σ i j ⊨ rinf) = (j = ∞)
using assms unfolding rinf-d-def by (simp add: rfinite-defs)

lemma wchop-defs[simp]:
assumes
       i ≠ ∞
       j ≠ −∞
       i ≤ j
shows  (σ i j ⊨ f wchop g) =
       ((∃ k. i ≤ eint k ∧ eint k ≤ j ∧ (σ i (eint k) ⊨ f) ∧ (σ (eint k) j ⊨ g)) ∨
        ((σ i j ⊨ f) ∧ j = ∞)
       )
using assms unfolding wchop-d-def by simp


lemma clinf-defs[simp]:
 assumes  i ≠ ∞
```

$j \neq -\infty$
$i \leq j$
shows $(\sigma\ i\ j\ c\models clinf) = (i = -\infty)$
using *assms* **unfolding** *clinf-d-def* **by** (*simp add*: *clfinite-defs*)

lemma *crinf-defs*[*simp*]:
assumes $i \neq \infty$
$j \neq -\infty$
$i \leq j$
shows $(\sigma\ i\ j\ c\models crinf) = (j = \infty)$
using *assms* **unfolding** *crinf-d-def* **by** (*simp add*: *crfinite-defs*)

lemma *cwchop-defs*[*simp*]:
assumes
$i \neq \infty$
$j \neq -\infty$
$i \leq j$
shows $(\sigma\ i\ j\ c\models f\ cwchop\ g) =$
$((\exists k.\ i \leq eint\ k \wedge eint\ k \leq j \wedge (\sigma\ i\ (eint\ k)\ c\models f) \wedge (\sigma\ (eint\ k)\ j\ c\models g)) \vee$
$((\sigma\ i\ j\ c\models f) \wedge j = \infty)$
$)$
using *assms* **unfolding** *cwchop-d-def* **by** *simp*

definition *agree-on* :: *interval* $\Rightarrow$ *interval* $\Rightarrow$ *eint* $\Rightarrow$ *eint* $\Rightarrow$ *bool*
where $agree\text{-}on\ \sigma\ \sigma'\ i\ j = (\forall k.\ i \leq k \wedge k \leq j \longrightarrow (\sigma\ k) = (\sigma'\ k))$

lemma *agree-on-defs*[*simp*]:
$agree\text{-}on\ \sigma\ \sigma'\ i\ j = (\forall k.\ i \leq k \wedge k \leq j \longrightarrow (\sigma\ k) = (\sigma'\ k))$
unfolding *agree-on-def* **by** *simp*

lemma *agree-on-sub*:
assumes $i \leq j$
$agree\text{-}on\ \sigma\ \sigma'\ i\ j$
$i \leq l0$
$l0 \leq l1$
$l1 \leq j$
shows $agree\text{-}on\ \sigma\ \sigma'\ l0\ l1$
using *assms*
using *order-trans* **by** *auto*

lemma *nridx-eint-gr-first*:
fixes i :: *eint*
assumes *nfinite l*
$i \neq -\infty$
$i \neq \infty$
$(nnth\ l\ 0) = i$
$nridx\ (<)\ l$
$(enat\ k) \leq nlength\ l$

shows $i \leq (nnth\ l\ k)$
using *assms*
nridx-gr-first[*of* $\lambda\ (a :: eint)\ b.\ a < b\ l\ k$]
using *order-le-less* **by** *auto*

lemma *nridx-eint-less-last*:
fixes $j :: eint$
assumes *nfinite l*
$j \neq -\infty$
$j \neq \infty$
$nlast\ l = j$
$nridx\ (<)\ l$
$(enat\ k) < nlength\ l$
shows $(nnth\ l\ (Suc\ k)) \leq j$
using *assms*
nridx-less-last[*of* $\lambda\ (a :: eint)\ b.\ a < b\ l\ k\ the\text{-}enat\ (nlength\ l)$]
apply *simp*
by (*metis Suc-lessI enat-ord-simps*(*2*) *less-eq-eint-def less-imp-Suc-add nfinite-conv-nlength-enat*
nle-le nnth-nlast nridx-less the-enat.simps transp-on-less)

lemma
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
$(\sigma\ i\ j\ c\models f)$
agree-on $\sigma\ \sigma'\ i\ j$
shows $(\sigma'\ i\ j\ c\models f)$
using *assms*
proof (*induction f arbitrary*: $\sigma\ \sigma'\ i\ j$)
case *cfalse-d*
then show *?case* **by** *simp*
next
case (*catom-d x*)
then show *?case* **apply** *simp*
by (*metis eint-infty-less*(*2*) *int-of-eint.simps*(*1*) *less-eint.elims*(*2*) *order-refl*)
next
case (*ciimp-d f1 f2*)
then show *?case* **apply** *simp* **by** *metis*
next
case (*cnext-d f*)
then show *?case*
proof −
have *c1*: $(\ \sigma'\ i\ j\ c\models cnext\ f) \longleftrightarrow (i \neq -\infty \wedge i < j \wedge (\sigma'\ (i + eint\ 1)\ j\ c\models f))$
by *simp*
have *c2*: $(i \neq -\infty \wedge i < j \wedge (\sigma\ (i + eint\ 1)\ j\ c\models f))$
using *cnext-d* **by** *simp*
have *c3*: $\forall k.\ i \leq eint\ k \wedge eint\ k \leq j \longrightarrow \sigma\ k = \sigma'\ k$
using *cnext-d* **by** *auto*

```
  have c4: agree-on σ σ' (i + eint 1) j
    unfolding agree-on-def
    by (metis MInfty-neq-eint(2) PInfty-neq-eint(1) add.right-neutral add-increasing2 c3 eint-0-plus
      eint-add-le-add-iff2 eint-less-eq(3) zero-less-one-class.zero-le-one)
  have c5: (i ≠ − ∞ ∧ i < j ∧ (σ (i + eint 1) j c⊨ f))
   using cnext-d.IH using c2 cnext-d.prems(4) by blast
  have c6: i + eint 1 ≤ j
    by (metis c5 cnext-d.prems(2) eint-less-le-alt int-of-eint.cases plus-eint.simps(1))
  have c7: i + eint 1 ≠ ∞
    by (simp add: cnext-d.prems(2))
  have c8:  (σ' (i + eint 1) j c⊨ f)
    using cnext-d.IH[of (i + eint 1)  j σ σ' ]
    using c4 c5 c6 c7 cnext-d.prems(3) by blast
  show ?thesis    by (simp add: c5 c8)
 qed
next
case (cchop-d f1 f2)
then show ?case
  proof −
   have 1:   (∃k. i ≤ eint k ∧ eint k ≤ j ∧ (σ i (eint k) c⊨ f1) ∧ (σ (eint k) j c⊨ f2))
     using cchop-d.prems(4) by auto
   obtain k where 2: i ≤ eint k ∧ eint k ≤ j ∧ (σ i (eint k) c⊨ f1) ∧ (σ (eint k) j c⊨ f2)
    using 1 by auto
   have 3: agree-on σ σ' i k
     by (meson 2 agree-on-defs cchop-d.prems(5) order-trans)
   have 4: agree-on σ σ' k j
     by (meson 2 agree-on-defs cchop-d.prems(5) order-trans)
   have 5: σ' i k c⊨ f1
      by (metis 2 3 MInfty-neq-eint(2) cchop-d.IH(1) cchop-d.prems(2))
   have 6: σ' k j c⊨ f2
     using 2 4 PInfty-neq-eint(1) cchop-d.IH(2) cchop-d.prems(3) by blast
   have 7: σ' i j c⊨ f1 cschop f2
    using 5 6 2 semantics-introspective-nl-itl.simps(5) by blast
   show ?thesis
   using 7 by blast
 qed
next
case (cchopstar-d f)
then show ?case
  proof (cases i=j)
  case True
  then show ?thesis
  using semantics-introspective-nl-itl.simps(6) by blast
  next
  case False
  then show ?thesis
    proof (cases i≠−∞ ∧ j≠ ∞)
    case True
    then show ?thesis
      proof −
```

```
  have 1: σ i j c⊨ cchopstar f
    using cchopstar-d.prems by blast
  have 2: (∃ (lrf:: eint nellist).
 (nnth lrf 0) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
  nridx (λ a b. a < b ∧ (σ a b c⊨ f)) lrf
)
    using ‹i≠−∞ ∧ j≠ ∞› ‹i ≠ j› cchopstar-d.prems by simp
  obtain lrf where 3: (nnth lrf 0) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
  nridx (λ a b. a < b ∧ (σ a b c⊨ f)) lrf
   using 2 by blast
  have 4: (∀ (k::nat) . ( (enat k)< (nlength lrf)) ⟶
                        ( σ (nnth lrf k) (nnth lrf (Suc k))   c⊨ f) )
    by (metis (no-types, lifting) 3 eSuc-enat ileI1 nridx-expand)
  have 5: ⋀k. ( (enat k)< (nlength lrf)) ⟶ (nnth lrf k) ≤ (nnth lrf (Suc k))
    by (metis (no-types, lifting) 3 eSuc-enat ileI1 nridx-expand order-less-imp-le)
  have 51: nridx (<) lrf
    by (metis (no-types, lifting) 3 nridx-expand)
  have 6: ⋀k. ( (enat k)< (nlength lrf)) ⟶ i ≤ (nnth lrf k)
     using 51 nridx-eint-gr-first
     by (simp add: 3 True cchopstar-d.prems(2))
  have 7: ⋀k. ( (enat k)< (nlength lrf)) ⟶ (nnth lrf (Suc k)) ≤ j
      using nridx-eint-less-last
      by (simp add: 3 51 True cchopstar-d.prems(3))
  have 8: ⋀k. ( (enat k)< (nlength lrf)) ⟶
            agree-on σ σ' (nnth lrf k) (nnth lrf (Suc k))
    using agree-on-sub[of i j σ σ']
    by (meson 5 6 7 cchopstar-d.prems(1) cchopstar-d.prems(5))
  have 9: ⋀k. ( (enat k)< (nlength lrf)) ⟶
            ( σ' (nnth lrf k) (nnth lrf (Suc k))   c⊨ f)
    by (metis 4 5 6 7 8 True cchopstar-d.IH eint-infty-less-eq(2) eint-less-eq(1) nle-le)
  have 10: nridx (λ a b. a < b ∧ (σ' a b c⊨ f)) lrf
    using 9
    by (metis (no-types, lifting) 3 Suc-ile-eq nridx-expand)
   show ?thesis
   using 10 3 True by auto
  qed
next
case False
then show ?thesis
  proof (cases i = −∞ ∧ j≠ ∞)
  case True
  then show ?thesis
    proof −
     have b1: (∃ (li :: eint nellist).
           (nnth li 0) = j ∧ ¬ nfinite li  ∧ nridx (λ a b. b < a ∧ (σ b a c⊨ f)) li
           )
     using ‹i = −∞ ∧ j≠ ∞› cchopstar-d.prems by simp
     obtain li where b2: (nnth li 0) = j ∧ ¬ nfinite li  ∧
                       nridx (λ a b. b < a ∧ (σ b a c⊨ f)) li
     using b1 by blast
```

have *b3*: (∀ (*k*::*nat*) .
(*σ* (*nnth li* (*Suc k*)) (*nnth li k*) *c*⊨ *f*))
by (*metis* (*no-types*, *lifting*) *b2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand*)
have *b5*: ⋀*k*. (*nnth li* (*Suc k*)) ≤(*nnth li k*)
by (*metis* (*no-types*, *lifting*) *b2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand order-less-imp-le*)
have *b51*: *nridx* (>) *li*
by (*metis* (*no-types*, *lifting*) *b2 nridx-expand*)
have *b8*: ⋀*k*. *agree-on σ σ′* (*nnth li* (*Suc k*)) (*nnth li k*)
by (*metis* (*mono-tags*, *lifting*) *b5 True agree-on-sub b2 cchopstar-d.prems*(5) *eint-infty-less-eq*(2) *lift-Suc-antimono-le nle-le zero-le*)
have *b9*: ⋀*k*.
(*σ′* (*nnth li* (*Suc k*)) (*nnth li k*) *c*⊨ *f*)
by (*metis* (*no-types*, *lifting*) *b5 b8 MInfty-neq-eint*(1) *PInfty-neq-eint*(1) *True b2 cchopstar-d.IH cchopstar-d.prems*(2) *enat-ile less-eint.elims*(2) *linorder-le-cases nfinite-conv-nlength-enat nridx-expand*)
have *b10*: *nridx* (λ *a b*. *a* > *b* ∧ (*σ′ b a c*⊨ *f*)) *li*
using *b9*
by (*metis* (*no-types*, *lifting*) *b2 nridx-expand*)
show *?thesis*
using *b10 True b2* **by** *auto*
qed
next
case *False*
then show *?thesis*
proof (*cases i* ≠ −∞ ∧ *j* = ∞)
case *True*
then show *?thesis*
proof −
have *c1*: (∃ (*ri* :: *eint nellist*).
(*nnth ri 0*) = *i* ∧ ¬ *nfinite ri* ∧ *nridx* (λ *a b*. *a* < *b* ∧ (*σ a b c*⊨ *f*)) *ri*
)
using ‹*i* ≠ −∞ ∧ *j* = ∞› *cchopstar-d.prems* **by** *simp*
obtain *ri* **where** *c2*: (*nnth ri 0*) = *i* ∧ ¬ *nfinite ri* ∧ *nridx* (λ *a b*. *a* < *b* ∧ (*σ a b c*⊨ *f*)) *ri*
using *c1* **by** *blast*
have *c3*: (∀ (*k*::*nat*) .
(*σ* (*nnth ri k*) (*nnth ri* (*Suc k*)) *c*⊨ *f*))
by (*metis* (*no-types*, *lifting*) *c2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand*)
have *c5*: ⋀*k*. (*nnth ri k*) ≤ (*nnth ri* (*Suc k*))
by (*metis* (*no-types*, *lifting*) *c2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand order-less-imp-le*)
have *c51*: *nridx* (<) *ri*
by (*metis* (*no-types*, *lifting*) *c2 nridx-expand*)

have *c6*: ⋀*k*. *i* ≤ (*nnth ri k*)
using *c51 nridx-eint-gr-first*
using *c2 c5 lift-Suc-mono-le* **by** *blast*
have *c8*: ⋀*k*. *agree-on σ σ′* (*nnth ri k*) (*nnth ri* (*Suc k*))
using *True agree-on-sub c2 c5 cchopstar-d.prems*(5) *eint-less-eq*(1) *lift-Suc-mono-le* **by** *blast*
have *c9*: ⋀*k*.
(*σ′* (*nnth ri k*) (*nnth ri* (*Suc k*)) *c*⊨ *f*)

by (*metis* (*no-types*, *lifting*) *PInfty-neq-eint*(*1*) *True c2 c5 c6 c8 cchopstar-d.IH eint-infty-less-eq*(*2*) *enat-ile less-eint.elims*(*2*) *linorder-le-cases nfinite-conv-nlength-enat nridx-expand*)
have *c10*: *nridx* (λ *a b*. *a* < *b* $\wedge$ (σ' *a b* *c*$\models$ *f*)) *ri*
using *c9*
by (*metis* (*no-types*, *lifting*) *c2 nridx-expand*)
show *?thesis*
using *True c10 c2* **by** *auto*
qed
next
case *False*
then show *?thesis*
proof −
have *d0*: *i* = −∞ $\wedge$ *j* = ∞
using ‹*i* ≠*j*› ‹¬(*i* ≠ −∞ $\wedge$ *j* = ∞)› ‹¬(*i* = −∞ $\wedge$ *j*≠ ∞)› ‹¬(*i*≠−∞ $\wedge$ *j*≠ ∞)›
by *blast*
have *d1*: ($\exists$ (*li* :: *int nellist*) (*ri* :: *int nellist*) .
(*nnth li 0*) = (*nnth ri 0*) $\wedge$
¬ *nfinite li* $\wedge$ *nridx* (λ *a b*. *b* < *a* $\wedge$ (σ *b a c*$\models$ *f*)) *li* $\wedge$
¬ *nfinite ri* $\wedge$ *nridx* (λ *a b*. *a* < *b* $\wedge$ (σ *a b c*$\models$ *f*)) *ri*
)
using *d0 cchopstar-d.prems* **by** *simp*
obtain *ri li* **where** *d2*: (*nnth* (*li* :: *int nellist*) *0*) = (*nnth* (*ri* :: *int nellist*) *0*) $\wedge$
¬ *nfinite li* $\wedge$ *nridx* (λ *a b*. *b* < *a* $\wedge$ (σ *b a c*$\models$ *f*)) *li* $\wedge$
¬ *nfinite ri* $\wedge$ *nridx* (λ *a b*. *a* < *b* $\wedge$ (σ *a b c*$\models$ *f*)) *ri*
using *d1* **by** *blast*
have *d3*: ($\forall$ (*k*::*nat*) .
(σ (*nnth ri k*) (*nnth ri* (*Suc k*)) *c*$\models$ *f*))
by (*metis* (*no-types*, *lifting*) *d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand*)
have *d4*: ($\forall$ (*k*::*nat*) .
(σ (*nnth li* (*Suc k*)) (*nnth li k*) *c*$\models$ *f*))
by (*metis* (*no-types*, *lifting*) *d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand*)
have *d5*: $\bigwedge$*k*. (*nnth ri k*) ≤ (*nnth ri* (*Suc k*))
by (*metis* (*no-types*, *lifting*) *d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand order-less-imp-le*)
have *d50*: $\bigwedge$*k*. (*nnth li* (*Suc k*)) ≤ (*nnth li k*)
by (*metis* (*no-types*, *lifting*) *d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand order-less-imp-le*)
have *d51*: *nridx* (<) *ri*
by (*metis* (*no-types*, *lifting*) *d2 nridx-expand*)
have *d52*: *nridx* (>) *li*
by (*metis* (*no-types*, *lifting*) *d2 nridx-expand*)
have *d8*: $\bigwedge$*k*. *agree-on* σ σ' (*nnth ri k*) (*nnth ri* (*Suc k*))
using *cchopstar-d.prems*(*5*) *d0* **by** *auto*
have *d80*: $\bigwedge$*k*. *agree-on* σ σ' (*nnth li* (*Suc k*)) (*nnth li k*)
using *cchopstar-d.prems*(*5*) *d0* **by** *auto*
have *d9*: $\bigwedge$*k*.
(σ' (*nnth ri k*) (*nnth ri* (*Suc k*)) *c*$\models$ *f*)
by (*meson MInfty-neq-eint*(*1*) *PInfty-neq-eint*(*1*) *cchopstar-d.IH d3 d5 d8 eint-less-eq*(*3*))
have *d90*: $\bigwedge$*k*.
(σ' (*nnth li* (*Suc k*)) (*nnth li k*) *c*$\models$ *f*)

```
          by (metis MInfty-neq-eint(1) PInfty-neq-eint(2) cchopstar-d.IH d4 d50 d80 eint-less-eq(3))
         have d10: nridx (λ a b. a < b ∧ (σ' a b   c⊨ f)) ri
          using d9 by (metis (no-types, lifting) d2 nridx-expand)
         have d11: nridx (λ a b. b < a ∧ (σ' b a   c⊨ f)) li
          using d90 by (metis (no-types, lifting) d2 nridx-expand)
        show ?thesis
        using  d10 d2 d11 d0 by auto
        qed
       qed
     qed
   qed
next
case (cexists-d x1 f)
then show ?case
   proof −
    have 1: (∃ l. (upd σ x1 l) i j c⊨ f )
    using cexist-sigma-cexist-value cexists-d.prems(4) by auto
    obtain l where 2: (upd σ x1 l) i j c⊨ f
     using 1 by blast
    have 3: (⋀i j σ σ'. i ≤ j ⟹ i ≠ ∞ ⟹ j ≠ − ∞ ⟹
              σ i j c⊨ f ⟹ agree-on σ σ' i j ⟹ σ' i j c⊨ f)
      using cexists-d.IH by blast
    have 4:  σ i j c⊨ cEx x1. f
        using cexists-d.prems by blast
    have 5: agree-on σ σ' i j
       using cexists-d.prems by blast
    have 6: agree-on (upd σ x1 l) (upd σ' x1 l) i j
        using 5 upd-def by force
    have 7:  (upd σ' x1 l) i j c⊨ f
      using 2 6 cexists-d.IH cexists-d.prems(1) cexists-d.prems(2) cexists-d.prems(3) by blast
    show ?thesis using 7 cexist-sigma-cexist-value by auto
  qed
qed


lemma schop-dist-ior:
 assumes  i ≤ j
        i ≠ ∞
        j ≠ −∞
 shows  (σ i j ⊨ ((f1 ior f2) schop f3) ieqv (f1 schop f3 ior f2 schop f3))
using assms by auto

lemma cschop-dist-ior:
 assumes  i ≤ j
        i ≠ ∞
        j ≠ −∞
 shows  (σ i j c⊨ ((f1 cior f2) cschop f3) cieqv (f1 cschop f3 cior f2 cschop f3))
using assms by auto
```

lemma *exists-atom*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models Ex\ n.\ \$n) = (i \neq -\infty)$
proof −
have *1*: $(\sigma\ i\ j \models Ex\ n.\ \$n) \longrightarrow (i \neq -\infty)$
by *simp*
have *2*: $(i \neq -\infty) \longrightarrow$
$n \in (upd\text{-}add\text{-}sigma\ \sigma\ (int\text{-}of\text{-}eint\ i)\ n)\ (int\text{-}of\text{-}eint\ i)$
$\wedge\ (similar\ \sigma\ n\ (upd\text{-}add\text{-}sigma\ \sigma\ (int\text{-}of\text{-}eint\ i)\ n))$ **apply** *simp*
by (*metis fun-upd-apply insert-Diff-if similar-defs-alt singletonI*)
have *3*: $(i \neq -\infty) \longrightarrow (\exists \sigma'.\ n \in \sigma'\ (int\text{-}of\text{-}eint\ i) \wedge similar\ \sigma\ n\ \sigma')$
using *2* **by** *blast*
have *4*: $(i \neq -\infty) \longrightarrow (\sigma\ i\ j \models Ex\ n.\ \ \$n)$
using *3*
by (*meson exist-sigma-exist-value semantics-nl-itl.simps*(*2*) *semantics-nl-itl.simps*(*9*))
show *?thesis* **using** *1 4* **by** *blast*
qed

lemma *cexists-catom*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models cEx\ n.\ \$\$n) = (i \neq -\infty)$
proof −
have *1*: $(\sigma\ i\ j\ c\models cEx\ n.\ \$\$n) \longrightarrow (i \neq -\infty)$
by *simp*
have *2*: $(i \neq -\infty) \longrightarrow$
$n \in (upd\text{-}add\text{-}sigma\ \sigma\ (int\text{-}of\text{-}eint\ i)\ n)\ (int\text{-}of\text{-}eint\ i)$
$\wedge\ (similar\ \sigma\ n\ (upd\text{-}add\text{-}sigma\ \sigma\ (int\text{-}of\text{-}eint\ i)\ n))$ **apply** *simp*
by (*metis fun-upd-apply insert-Diff1 similar-defs-alt singletonI*)
have *3*: $(i \neq -\infty) \longrightarrow (\exists \sigma'.\ n \in \sigma'\ (int\text{-}of\text{-}eint\ i) \wedge similar\ \sigma\ n\ \sigma')$
using *2* **by** *blast*
have *4*: $(i \neq -\infty) \longrightarrow (\sigma\ i\ j\ c\models cEx\ n.\ \ \$\$n)$
using *3*
by (*meson cexist-sigma-cexist-value semantics-introspective-nl-itl.simps*(*2*) *semantics-introspective-nl-itl.simps*(*7*))
show *?thesis* **using** *1 4* **by** *blast*
qed

lemma *exists-not-atom*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models Ex\ n.\ inot\ \$n)$
using *assms* **apply** *simp* **unfolding** *upd-def*
by *auto*

lemma *cexists-not-catom*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models\ cEx\ n.\ cinot\ \$\$n)$
using *assms* **apply** *simp* **unfolding** *upd-def*
by *auto*

lemma *exists-exists*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models Ex\ n.\ (Ex\ n.\ f)) = (\sigma\ i\ j \models Ex\ n.\ f)$
using *assms*
apply *simp*
using *similar-refl similar-trans*
using *upd-absorb* **by** *presburger*

lemma *cexists-cexists*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models\ cEx\ n.\ (cEx\ n.\ f)) = (\sigma\ i\ j\ c\models\ cEx\ n.\ f)$
using *assms*
apply *simp*
using *upd-absorb similar-refl similar-trans* **by** *presburger*

lemma *exists-ior-dist*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models (Ex\ n.\ f1\ ior\ f2)) = (\sigma\ i\ j \models (Ex\ n.\ f1)\ ior\ (Ex\ n.\ f2))$
using *assms* **by** *auto*

lemma *cexists-cior-dist*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models (cEx\ n.\ f1\ cior\ f2)) = (\sigma\ i\ j\ c\models (cEx\ n.\ f1)\ cior\ (cEx\ n.\ f2))$
using *assms* **by** *auto*

lemma *exists-schop-imp*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$

shows $(\sigma\ i\ j \models (Ex\ n.\ (f1\ schop\ f2))) \Longrightarrow (\sigma\ i\ j \models (Ex\ n.\ f1)\ schop\ (Ex\ n.\ f2))$
using *assms* **by** *auto*

lemma *cexists-cschop-imp*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j\ c\models (cEx\ n.\ (f1\ cschop\ f2))) \Longrightarrow (\sigma\ i\ j\ c\models (cEx\ n.\ f1)\ cschop\ (cEx\ n.\ f2))$
using *assms* **by** *auto*

lemma *sfrdiamond-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models rdiamond\ (inot(skip\ schop\ (inot\ f)))) =$
$(\ j <\ \infty \wedge (\exists k.\ k = j \vee (k > j \wedge (\sigma\ j\ k \models f))))$
using *assms* **apply** *auto*
apply (*metis dual-order.refl less-le-not-le semantics-nl-itl.simps*(*4*) *skip-d-def*)
by (*metis MInfty-neq-eint*(*1*) *not-less-iff-gr-or-eq order-le-less skip-defs*)

lemma *spldiamond-defs*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models ldiamond\ (inot\ ((inot\ f)\ schop\ skip))) =$
$(\ i > -\infty \wedge (\exists k.\ \ k = i \vee (k < i \wedge (\sigma\ k\ i \models f))))$
using *assms* **apply** *auto*
apply *fastforce*
using *add.commute* **by** *auto*

lemma *strictly-future-eqv-future-sem*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $\exists\ f.\ \ (\sigma\ i\ j\ sf\models sfrdiamond\ g) = (\sigma\ i\ j\ f\models f)$
using *assms*
proof (*induction g arbitrary*: j)
case (*fintro-d x*)
then show *?case* **apply** *auto*
by (*metis semantics-future-nl-itl.simps*(*2*))
next
case (*finot-d g*)
then show *?case* **by** *auto*
next
case (*fior-d g1 g2*)
then show *?case* **apply** *auto*
by *metis*
next
case (*frdiamond-d g*)

then show *?case* **by** *auto*
qed

lemma *strictly-past-eqv-past-sem*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $\exists\ f.\ (\sigma\ i\ j\ sp\models spldiamond\ g) = (\sigma\ i\ j\ p\models f)$
using *assms*
proof (*induction g arbitrary*: *i*)
case (*pintro-d x*)
then show *?case* **apply** *auto*
by (*metis semantics-past-nl-itl.simps*(*2*))
next
case (*pinot-d g*)
then show *?case* **by** *auto*
next
case (*pior-d g1 g2*)
then show *?case* **apply** *auto* **by** *metis*
next
case (*pldiamond-d g*)
then show *?case* **by** *auto*
qed

lemma *state-nl-itl-list-defs*:
state-nl-itl-list L $\longleftrightarrow$ ($\forall$ *i*<*length L*. *state-nl-itl* (*L*!*i*))
proof (*induct L*)
case *Nil*
then show *?case* **unfolding** *state-nl-itl-list-def* **by** *simp*
next
case (*Cons a L*)
then show *?case*
proof −
have *1*: *state-nl-itl-list* (*a* # *L*) $\longleftrightarrow$ *state-nl-itl a* $\wedge$ *state-nl-itl-list L*
unfolding *state-nl-itl-list-def* **by** *simp*
have *2*: ($\forall$ *i*<*length* (*a*#*L*). *state-nl-itl* ((*a*#*L*)!*i*)) $\longleftrightarrow$
state-nl-itl ((*a*#*L*)!*0*) $\wedge$
($\forall$ *i*. *0*<*i* $\wedge$ *i*<*length* (*a*#*L*)$\longrightarrow$ *state-nl-itl* ((*a*#*L*)!*i*))
by (*metis length-greater-0-conv list.discI zero-less-iff-neq-zero*)
have *3*: *state-nl-itl* ((*a*#*L*)!*0*) $\longleftrightarrow$ *state-nl-itl a*
by *simp*
have *4*: ($\forall$ *i*. *0*<*i* $\wedge$ *i*<*length* (*a*#*L*)$\longrightarrow$ *state-nl-itl* ((*a*#*L*)!*i*)) $\longleftrightarrow$
($\forall$ *i*<*length* (*L*). *state-nl-itl* (*L*!*i*))
by *auto*
show *?thesis*
using *1 2 3 4 local.Cons* **by** *presburger*
qed
qed

lemma *big-ior-defs* :
(σ i j ⊨ (*big-ior* F)) ⟷ (∃ k< *length* F. (σ i j ⊨ F ! k))
proof (*induct* F)
case *Nil*
then show *?case* **unfolding** *big-ior-def* **by** *simp*
next
case (*Cons* a F)
then show *?case*
proof −
have *1*: (σ i j ⊨ (*big-ior* (a#F))) ⟷ (σ i j ⊨ a *ior* (*big-ior* F))
unfolding *big-ior-def* **by** *simp*
have *2*: (∃ k<*length* (a # F). σ i j ⊨ (a # F) ! k) ⟷
(σ i j ⊨ (a # F) ! 0) ∨
(∃ k. 0 < k ∧ k<*length* (a # F) ∧ (σ i j ⊨ (a # F) ! k))
by (*metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero*)
have *3*: (σ i j ⊨ (a # F) ! 0) = (σ i j⊨ a)
by *auto*
have *4*: (∃ k. 0 < k ∧ k<*length* (a # F) ∧ (σ i j ⊨ (a # F) ! k)) ⟷
(∃ k<*length* (F). (σ i j ⊨ (F) ! k))
by *auto*
(*metis Suc-le-eq diff-Suc-Suc diff-zero gr0-conv-Suc less-Suc-eq-le*)
show *?thesis*
using *1 2 4 local.Cons* **by** *force*
qed
qed

lemma *big-iand-defs* :
(σ i j ⊨ (*big-iand* F)) ⟷ (∀ k< *length* F. (σ i j ⊨ F ! k))
proof (*induct* F)
case *Nil*
then show *?case* **unfolding** *big-iand-def* **by** *simp*
next
case (*Cons* a F)
then show *?case*
proof −
have *1*: (σ i j ⊨ (*big-iand* (a#F))) ⟷ (σ i j ⊨ a *iand* (*big-iand* F))
unfolding *big-iand-def* **by** *simp*
have *2*: (∀ k<*length* (a # F). σ i j ⊨ (a # F) ! k) ⟷
(σ i j ⊨ (a # F) ! 0) ∧
(∀ k. 0 < k ∧ k<*length* (a # F) ⟶ (σ i j ⊨ (a # F) ! k))
by (*metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero*)
have *3*: (σ i j ⊨ (a # F) ! 0) = (σ i j ⊨ a)
by *auto*
have *4*: (∀ k. 0 < k ∧ k<*length* (a # F) ⟶ (σ i j ⊨ (a # F) ! k)) ⟷
(∀ k<*length* (F). (σ i j ⊨ (F) ! k))
by *auto*
show *?thesis*

using *1 2 4 local.Cons* **by** *force*
qed
qed

lemma *big-ior-map-iand-zip* :
assumes *length F* =*length G*
shows (*σ i j* ⊨ *big-ior* (*map2 iand-d F G*)) ⟷
(∃ *k* < *length F*. (*σ i j* ⊨ *F* ! *k iand G* ! *k*))
using *assms*
proof (*induct F arbitrary*: *G*)
case *Nil*
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*Cons a F*)
then show *?case*
proof (*cases G*=[])
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** (*auto simp add*: *big-ior-defs*)
qed
qed

lemma *big-iand-map-iimp-zip* :
assumes *length F* =*length G*
shows (*σ i j* ⊨ *big-iand* (*map2 iimp-d F G*)) ⟷
(∀ *k* < *length F*. (*σ i j* ⊨ *F* ! *k iimp G* ! *k*))
using *assms*
proof (*induct F arbitrary*: *G*)
case *Nil*
then show *?case* **by** (*simp add*: *big-iand-defs*)
next
case (*Cons a F*)
then show *?case*
proof (*cases G*=[])
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** (*auto simp add*: *big-iand-defs*)
qed
qed

lemma *map2-iand-defs* :
assumes *length F* = *length G*

shows ($\exists\, k{<}$*length G*. σ *i j* $\models$ *map2 iand-d F G* ! *k*) $\longleftrightarrow$
($\exists\, k{<}$*length G*. (σ *i j* $\models$ *F* !*k*) $\wedge$ (σ *i j* $\models$ *G* ! *k*))
using *assms*
proof (*induct G arbitrary*: *F*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a G*)
then show *?case*
proof (*cases F*=[])
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** *auto*
qed
qed

lemma *map2-iand-inot-defs* :
assumes *length F* = *length G*
shows ($\exists\, k{<}$*length G*. σ *i j* $\models$ *map2* ($\lambda x\ y.\ x$ *iand inot y*) *F G* ! *k*) $\longleftrightarrow$
($\exists\, k{<}$*length G*. (σ *i j* $\models$ *F*! *k*) $\wedge$ $\neg$(σ *i j* $\models$ *G*!*k*))
using *assms*
proof (*induct G arbitrary*: *F*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a G*)
then show *?case*
proof (*cases F*=[])
case *True*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case *False*
then show *?thesis* **using** *Cons* **by** *auto*
qed
qed

7.3 Validity

definition *valid* :: *nl-itl* $\Rightarrow$ *bool* (($\vdash$ -) 5)
where ($\vdash$ *f*) = ($\forall$ σ *i j*. *i* $\leq$ *j* $\wedge$ *i* $\neq \infty \wedge$ *j* $\neq -\infty \longrightarrow$($\sigma$ *i j* $\models$ *f*))

lemma *itl-valid* [*simp*] :
($\vdash$ *f*) = ($\forall$ σ *i j*. *i* $\leq$ *j* $\wedge$ *i* $\neq \infty \wedge$ *j* $\neq -\infty \longrightarrow$ (σ *i j* $\models$ *f*))
by (*simp add*: *valid-def*)

lemma *itl-ieq*:

$(\vdash f \; ieqv \; g) = (\forall \; \sigma \; i \; j.\; i \leq j \wedge i \neq \infty \wedge j \neq -\infty \longrightarrow (\sigma \; i \; j \models f) = (\sigma \; i \; j \models g))$
by (*auto simp add*: *ieqv-d-def iand-d-def ior-d-def inot-d-def*)

lemma *big-ior-iimp-subset-eq*:
assumes *set L1* $\subseteq$ *set L2*
shows $\vdash$ *big-ior L1 iimp big-ior L2*
using *assms*
proof (*induct rule*: *list-induct2'*)
case *1*
then show *?case* **by** *simp*
next
case (*2 x xs*)
then show *?case* **by** *simp*
next
case (*3 y ys*)
then show *?case* **by** (*simp add*: *big-ior-defs*)
next
case (*4 x xs y ys*)
then show *?case*
proof (*cases x=y*)
case *True*
then show *?thesis* **using** *4* **apply** (*auto simp add*: *big-ior-defs*)
by (*metis diff-Suc-1 in-set-conv-nth length-Cons less-Suc-eq-0-disj list.simps*(*15*) *nth-Cons' subsetD*)
next
case *False*
then show *?thesis* **using** *4* **apply** (*auto simp add*: *big-ior-defs*)
by (*metis 4.prems in-set-conv-nth length-Cons subsetD*)
qed
qed

lemma *big-ior-ieqv-set-eq*:
assumes *set L1* = *set L2*
shows $\vdash$ *big-ior L1 ieqv big-ior L2*
using *assms*
by (*metis big-ior-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-ior-ieqv-remdups*:
shows $\vdash$ *big-ior L ieqv big-ior* (*remdups L*)
by (*metis big-ior-ieqv-set-eq set-remdups*)

lemma *big-iand-iimp-subset-eq*:
assumes *set L1* $\subseteq$ *set L2*
shows $\vdash$ *big-iand L2 iimp big-iand L1*
using *assms*
proof (*induct rule*: *list-induct2'*)

case *1*
then show *?case* **by** *simp*
next
case (*2 x xs*)
then show *?case* **by** (*simp add*: *big-iand-defs*)
next
case (*3 y ys*)
then show *?case* **by** *simp*
next
case (*4 x xs y ys*)
then show *?case*
proof (*cases x=y*)
case *True*
then show *?thesis* **using** *4* **apply** (*auto simp add*: *big-iand-defs*)
using *4 all-nth-imp-all-set length-Cons subsetD*
by (*metis nth-mem*)
next
case *False*
then show *?thesis* **using** *4* **apply** (*auto simp add*: *big-iand-defs*)
by (*metis 4.prems all-nth-imp-all-set length-Cons nth-mem subsetD*)
qed
qed

lemma *big-iand-ieqv-set-eq*:
assumes *set L1 = set L2*
shows $\vdash$ *big-iand L1 ieqv big-iand L2*
using *assms*
by (*metis big-iand-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-iand-ieqv-remdups*:
$\vdash$ *big-iand L ieqv big-iand* (*remdups L*)
by (*metis big-iand-ieqv-set-eq set-remdups*)

lemma *linf-subst*:
assumes $n \notin$ *bvars f*
$\bigwedge z$. $z \in$ *fvars w* $\Longrightarrow$ $z \notin$ *bvars f*
state-nl-itl w
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows (σ *i j* $\models$ *suform linf n w*) = ((*upd* σ *n* (λ *c*. (σ *c c* $\models$ *w*))) *i j* $\models$ *linf*)
using *assms*
unfolding *linf-d-def iand-d-def lfinite-d-def itrue-d-def empty-d-def more-d-def ior-d-def inot-d-def*
by *simp*

lemma *linf-subst-1*:
assumes $n \notin$ *bvars f*

$\bigwedge z\ .\ z \in fvars\ w \Longrightarrow z \notin bvars\ f$
$state\text{-}nl\text{-}itl\ w$
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models\ suform\ linf\ n\ w) =\ (\sigma\ i\ j \models linf)$
using *assms*
unfolding *linf-d-def iand-d-def lfinite-d-def itrue-d-def empty-d-def more-d-def ior-d-def inot-d-def*
by *simp*

lemma *state-nl*:
assumes *state-nl-itl w*
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
$(\sigma\ i\ j \models \$x)$
shows $i \neq -\infty$
using *assms* **by** *simp*

lemma *lfinite-subst-2*:
assumes $n \notin bvars\ f$
$\bigwedge z\ .\ z \in fvars\ w \Longrightarrow z \notin bvars\ f$
$state\text{-}nl\text{-}itl\ w$
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models\ suform\ (\$x)\ n\ (w\ iand\ lfinite)) =\ ((upd\ \sigma\ n\ (\lambda\ c.\ (\sigma\ c\ c \models w))\ )\ )\ \ i\ j \models \$x\ )$
proof −
have *1*: $(\sigma\ i\ j \models suform\ (\$x)\ n\ (w\ iand\ lfinite)) =$
$(if\ x = n\ then\ (\sigma\ i\ j \models (w\ iand\ lfinite))\ else\ (i \neq -\infty \wedge x \in \sigma\ (int\text{-}of\text{-}eint\ i)))$
by *simp*
have *15*: $(if\ x = n\ then\ (\sigma\ i\ j \models (w\ iand\ lfinite))\ else\ (i \neq -\infty \wedge x \in \sigma\ (int\text{-}of\text{-}eint\ i))) =$
$(\ i \neq -\infty \wedge (if\ x = n\ then\ (\sigma\ i\ j \models (w\ ))\ else\ (\ x \in \sigma\ (int\text{-}of\text{-}eint\ i))))$
using *assms(4) assms(5) assms(6) lfinite-defs* **by** *force*
have *2*: $((upd\ \sigma\ n\ (\lambda\ c.\ (\sigma\ c\ c \models w))\ )\ )\ \ i\ j \models \$x\ ) =$
$(i \neq -\infty \wedge x \in (upd\ \sigma\ n\ (\lambda x.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ (int\text{-}of\text{-}eint\ i))$
by *simp*
have *3*: $i \neq -\infty \Longrightarrow$
$x \in (upd\ \sigma\ n\ (\lambda x.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ (int\text{-}of\text{-}eint\ i) =$
$(if\ x = n\ then\ (\sigma\ i\ j \models (w\ ))\ else\ (\ x \in \sigma\ (int\text{-}of\text{-}eint\ i)))$

unfolding *upd-def* **apply** *simp*
by (*metis PInfty-neq-eint(1) abs-eint.elims assms(3) assms(5) assms(6) eint-int state-nl-itl-defs*)
show *?thesis*
using *1 15 2 3* **by** *blast*
qed

lemma *subst-suform*:

assumes $n \notin bvars\ f$
$\bigwedge z\ .\ z \in fvars\ w \Longrightarrow z \notin bvars\ f$
state-nl-itl w
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models\ suform\ f\ n\ (w\ iand\ lfinite)) = ((upd\ \sigma\ n\ (\lambda\ c.\ (\sigma\ c\ c \models w))\ )\ )\ \ i\ j \models f\ )$
using *assms*
proof (*induction f arbitrary*: $\sigma\ w\ i\ j\ n$)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case*
using *lfinite-subst-2* **by** *blast*
next
case (*iimp-d f1 f2*)
then show *?case*
by *simp*
next
case (*next-d f*)
then show *?case*
proof −
have *n1*: $(\sigma\ i\ j \models suform\ (next\ f)\ n\ (w\ iand\ lfinite)) =$
$(i \neq -\infty \wedge i < j \wedge (\sigma\ (i + eint\ 1)\ j \models suform\ f\ n\ (w\ iand\ lfinite)))$
by *simp*
have *n2*: $((upd\ \sigma\ n\ (\lambda x.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ i\ j \models next\ f) =$
$(i \neq -\infty \wedge i < j \wedge ((upd\ \sigma\ n\ (\lambda x.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ (i + eint\ 1)\ j \models f))$
by *simp*
have *n3*: $i \neq -\infty \wedge i < j \Longrightarrow i + eint\ 1 \leq j$
by (*metis eint-less-le-alt int-of-eint.cases next-d.prems(5) plus-eint.simps(1)*)
have *n4*: $i \neq -\infty \wedge i < j \Longrightarrow$
$(\sigma\ (i + eint\ 1)\ j \models suform\ f\ n\ (w\ iand\ lfinite)) =$
$((upd\ \sigma\ n\ (\lambda x.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ (i + eint\ 1)\ j \models f)$
using *next-d.IH*[*of n w* $(i + eint\ 1)\ j\ \sigma$]
using *n3 next-d.prems(1) next-d.prems(2) next-d.prems(3) next-d.prems(5) next-d.prems(6)* **by** *auto*
show *?thesis*
using *n1 n2 n4* **by** *blast*
qed
next
case (*chop-d f1 f2*)
then show *?case* **by** *auto*
next
case (*ldiamond-d f*)
then show *?case*
proof −
have *l1*: $(\sigma\ i\ j \models suform\ (ldiamond\ f)\ n\ (w\ iand\ lfinite)) =$
$(i \neq -\infty \wedge (\exists k{\leq}i.\ \sigma\ k\ i \models suform\ f\ n\ (w\ iand\ lfinite)))$
by *simp*
have *l2*: $((upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ i\ j \models ldiamond\ f) =$

$(i \neq -\infty \wedge (\exists\, k{\leq}i.\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ k\ i \models f))$

by *simp*
have *l3*: $i \neq -\infty \Longrightarrow$
$(\exists\, k{\leq}i.\ (\sigma\ k\ i \models suform\ f\ n\ (w\ iand\ lfinite))) =$
$(\exists\, k{\leq}i.\ ((upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ k\ i \models f))$
using *ldiamond-d.IH*[*of n w - i* σ]
using *bvars.simps*(*6*) *eint-infty-less-eq*(*1*) *ldiamond-d.prems*(*1*) *ldiamond-d.prems*(*2*) *ldiamond-d.prems*(*3*) *ldiamond-d.prems*(*5*) **by** *blast*
show *?thesis*
using *l1 l2 l3* **by** *blast*
qed
next
case (*rdiamond-d f*)
then show *?case*
proof −
have *r1*: $(\sigma\ i\ j \models suform\ (rdiamond\ f)\ n\ (w\ iand\ lfinite)) =$
$(j \neq \infty \wedge (\exists\, k{\geq}j.\ \sigma\ j\ k \models suform\ f\ n\ (w\ iand\ lfinite)))$
by *simp*
have *r2*: $((upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ i\ j \models rdiamond\ f) =$
$(j \neq \infty \wedge (\exists\, k{\geq}j.\ (upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ j\ k \models f))$
by *simp*
have *r3*: $j \neq \infty \Longrightarrow$
$(\exists\, k{\geq}j.\ \sigma\ j\ k \models suform\ f\ n\ (w\ iand\ lfinite)) =$
$(\exists\, k{\geq}j.\ (upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ j\ k \models f)$
using *rdiamond-d.IH*[*of n w j -* σ]
using *bvars.simps*(*7*) *eint-infty-less-eq*(*2*) *rdiamond-d.prems*(*1*) *rdiamond-d.prems*(*2*) *rdiamond-d.prems*(*3*) *rdiamond-d.prems*(*6*) **by** *blast*
show *?thesis*
using *r1 r2 r3* **by** *blast*
qed
next
case (*chopstar-d f*)
then show *?case*
proof −
have *c1*: $(\sigma\ i\ j \models suform\ (chopstar\ f\ )\ n\ (w\ iand\ lfinite)) =$
$(i = j \vee$
$i \neq j \wedge i \neq -\infty \wedge j \neq \infty \wedge$
$(\exists\, lrf{::}eint\ nellist.\ nnth\ lrf\ (0{::}nat) = i \wedge nfinite\ lrf \wedge nlast\ lrf = j \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ f\ n\ (w\ iand\ lfinite)))\ lrf) \vee$
$i \neq j \wedge i = -\infty \wedge j \neq \infty \wedge$
$(\exists\, li{::}eint\ nellist.\ nnth\ li\ (0{::}nat) = j \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ b < a \wedge (\sigma\ b\ a \models suform\ f\ n\ (w\ iand\ lfinite)))\ li) \vee$
$i \neq j \wedge i \neq -\infty \wedge j = \infty \wedge$
$(\exists\, ri{::}eint\ nellist.\ nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ f\ n\ (w\ iand\ lfinite)))\ ri) \vee$
$i \neq j \wedge i = -\infty \wedge j = \infty \wedge$
$(\exists\, (li{::}int\ nellist)\ ri{::}int\ nellist.$
$nnth\ li\ (0{::}nat) = nnth\ ri\ (0{::}nat) \wedge$
$\neg\ nfinite\ li \wedge$

```
      nridx (λ(a::int) b::int. b < a ∧ (σ (eint b) (eint a) ⊨ suform f n (w iand lfinite))) li ∧
     ¬ nfinite ri ∧ nridx (λ(a::int) b::int. a < b ∧ (σ (eint a) (eint b) ⊨ suform f n (w iand lfinite))) ri))
  by simp
 have c2: ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) i j ⊨ chopstar f ) =
        (i = j ∨
   i ≠ j ∧ i ≠ − ∞ ∧ j ≠ ∞ ∧
   (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
     nridx (λ(a::eint) b::eint. a < b ∧
          ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) lrf) ∨
   i ≠ j ∧ i = − ∞ ∧ j ≠ ∞ ∧
    (∃ li::eint nellist. nnth li (0::nat) = j ∧ ¬ nfinite li ∧
     nridx (λ(a::eint) b::eint. b < a ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) b a ⊨ f)) li) ∨
   i ≠ j ∧ i ≠ − ∞ ∧ j = ∞ ∧
    (∃ ri::eint nellist. nnth ri (0::nat) = i ∧ ¬ nfinite ri ∧
     nridx (λ(a::eint) b::eint. a < b ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) ri) ∨
   i ≠ j ∧ i = − ∞ ∧ j = ∞ ∧
   (∃ (li::int nellist) ri::int nellist.
      nnth li (0::nat) = nnth ri (0::nat) ∧
      ¬ nfinite li ∧
      nridx (λ(a::int) b::int. b < a ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint b) (eint a) ⊨ f)) li ∧
      ¬ nfinite ri ∧
      nridx (λ(a::int) b::int. a < b ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint a) (eint b) ⊨ f)) ri))
   by simp
 have c3: i = j ⟹ ?thesis
   using c1 c2 by presburger
 have c4: i≠j ⟹ i ≠ − ∞ ∧ j ≠ ∞ ⟹
        (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ (σ a b ⊨ suform f n (w iand lfinite))) lrf) ⟹
        (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) lrf)
  proof −
   assume c4a0: i≠j
   assume c4a1: i ≠ − ∞ ∧ j ≠ ∞
   assume c4a2: (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ (σ a b ⊨ suform f n (w iand lfinite))) lrf)
   show (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) lrf)
   proof −
    obtain lrf where c41: nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ (σ a b ⊨ suform f n (w iand lfinite))) lrf
    using c4a2 by blast
    have c42: nridx (λ(a::eint) b::eint. a < b ∧ ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) lrf

     using chopstar-d.IH[of n w - - σ]  c41 chopstar-d.prems
   by (metis (no-types, lifting) bvars.simps(8) eint-minus-less-minus eint-uminus-eq-reorder less-eint.simps(2)
nridx-expand order-less-imp-le)
```

```
    show ?thesis
    using c41 c42 by blast
   qed
  qed
 have c5: i≠j ⟹ i ≠ − ∞ ∧ j ≠ ∞ ⟹
        (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
         nridx (λ(a::eint) b::eint. a < b ∧ ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) lrf) ⟹
        (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ (σ a b ⊨ suform f n (w iand lfinite))) lrf)
 proof −
  assume c5a0: i≠j
  assume c5a1: i ≠ − ∞ ∧ j ≠ ∞
  assume c5a2: (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) lrf)
  show (∃ lrf::eint nellist. nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ (σ a b ⊨ suform f n (w iand lfinite))) lrf)
  proof −
   obtain lrf where c51: nnth lrf (0::nat) = i ∧ nfinite lrf ∧ nlast lrf = j ∧
          nridx (λ(a::eint) b::eint. a < b ∧ ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) a b ⊨ f)) lrf
    using c5a2 by blast
   have c52: nridx (λ(a::eint) b::eint. a < b ∧ (σ a b ⊨ suform f n (w iand lfinite))) lrf
     using chopstar-d.IH[of n w - - σ] c51 chopstar-d.prems
      by (metis (no-types, lifting) bvars.simps(8) less-eint.simps(2) less-eint.simps(3) nridx-expand or-
der-less-imp-le)
   show ?thesis using c51 c52 by blast
  qed
 qed
 have c6: i≠j ∧ i ≠ − ∞ ∧ j ≠ ∞ ⟹ ?thesis
   using c4 c5 apply simp by argo
 have c7: i≠j ⟹ i = − ∞ ∧ j ≠ ∞ ⟹
        (∃ li::eint nellist. nnth li (0::nat) = j ∧ ¬ nfinite li ∧
          nridx (λ(a::eint) b::eint. b < a ∧ (σ b a ⊨ suform f n (w iand lfinite))) li) ⟹
        (∃ li::eint nellist. nnth li (0::nat) = j ∧ ¬ nfinite li ∧
           nridx (λ(a::eint) b::eint. b < a ∧
            ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) b a ⊨ f)) li)
 proof −
  assume c7a0: i≠j
  assume c7a1: i = − ∞ ∧ j ≠ ∞
  assume c7a2: (∃ li::eint nellist. nnth li (0::nat) = j ∧ ¬ nfinite li ∧
          nridx (λ(a::eint) b::eint. b < a ∧ (σ b a ⊨ suform f n (w iand lfinite))) li)
  show (∃ li::eint nellist. nnth li (0::nat) = j ∧ ¬ nfinite li ∧
           nridx (λ(a::eint) b::eint. b < a ∧
            ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) b a ⊨ f)) li)
  proof −
   obtain li where c71: nnth li (0::nat) = j ∧ ¬ nfinite li ∧
          nridx (λ(a::eint) b::eint. b < a ∧ (σ b a ⊨ suform f n (w iand lfinite))) li
    using c7a2 by blast
   have c72: nridx (λ(a::eint) b::eint. b < a ∧
            ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) b a ⊨ f)) li
         using chopstar-d.IH[of n w - - σ] c71 chopstar-d.prems
```

by (*metis* (*no-types*, *lifting*) *bvars.simps*(*8*) *dual-order.strict-iff-order eint-minus-less-minus eint-uminus-uminus less-eint.simps*(*2*) *nridx-expand*)
show *?thesis* **using** *c71 c72* **by** *blast*
qed
qed
have *c8*: $i{\neq}j \Longrightarrow i = -\infty \wedge j \neq \infty \Longrightarrow$
$(\exists\, li{::}eint\ nellist.\ nnth\ li\ (0{::}nat) = j \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ b < a \wedge$
$((upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ b\ a \models f))\ li) \Longrightarrow$
$(\exists\, li{::}eint\ nellist.\ nnth\ li\ (0{::}nat) = j \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ b < a \wedge (\sigma\ b\ a \models suform\ f\ n\ (w\ iand\ lfinite)))\ li)$
proof −
assume *c8a0*: $i{\neq}j$
assume *c8a1*: $i = -\infty \wedge j \neq \infty$
assume *c8a2*: $(\exists\, li{::}eint\ nellist.\ nnth\ li\ (0{::}nat) = j \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ b < a \wedge$
$((upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ b\ a \models f))\ li)$
show $(\exists\, li{::}eint\ nellist.\ nnth\ li\ (0{::}nat) = j \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ b < a \wedge (\sigma\ b\ a \models suform\ f\ n\ (w\ iand\ lfinite)))\ li)$
proof −
obtain *li* **where** *c81*: $nnth\ li\ (0{::}nat) = j \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ b < a \wedge$
$((upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ b\ a \models f))\ li$
using *c8a2* **by** *blast*
have *c82*: $nridx\ (\lambda(a{::}eint)\ b{::}eint.\ b < a \wedge (\sigma\ b\ a \models suform\ f\ n\ (w\ iand\ lfinite)))\ li$
using *chopstar-d.IH*[*of n w - - σ*] *c81 chopstar-d.prems*
by (*metis* (*no-types*, *lifting*) *bvars.simps*(*8*) *less-eint.simps*(*2*) *less-eint.simps*(*3*) *nridx-expand order-less-imp-le*)
show *?thesis* **using** *c81 c82* **by** *blast*
qed
qed
have *c9*: $i{\neq}j \Longrightarrow i = -\infty \wedge j \neq \infty \Longrightarrow$ *?thesis*
using *c7 c8* **apply** *simp* **by** *argo*
have *c10*: $i{\neq}j \Longrightarrow i \neq -\infty \wedge j = \infty \Longrightarrow$
$(\exists\, ri{::}eint\ nellist.\ nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ f\ n\ (w\ iand\ lfinite)))\ ri) \Longrightarrow$
$(\exists\, ri{::}eint\ nellist.\ nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge$
$((upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ a\ b \models f))\ ri)$
proof −
assume *c10a0*: $i{\neq}j$
assume *c10a1*: $i \neq -\infty \wedge j = \infty$
assume *c10a2*: $(\exists\, ri{::}eint\ nellist.\ nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ f\ n\ (w\ iand\ lfinite)))\ ri)$
show $(\exists\, ri{::}eint\ nellist.\ nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge$
$((upd\ \sigma\ n\ (\lambda x{::}int.\ \sigma\ (eint\ x)\ (eint\ x) \models w))\ a\ b \models f))\ ri)$
proof −
obtain *ri* **where** *c101*: $nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ f\ n\ (w\ iand\ lfinite)))\ ri$

using *c10a2* **by** *blast*
have *c102*: *nridx* (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧
((*upd* σ *n* (λ*x*::*int*. σ (*eint x*) (*eint x*) ⊨ *w*)) *a b* ⊨ *f*)) *ri*
using *chopstar-d.IH*[*of n w - - σ*] *c101 chopstar-d.prems*
by (*metis* (*no-types*, *lifting*) *bvars.simps*(*8*) *less-eint.simps*(*2*) *less-eint.simps*(*3*) *nridx-expand order-less-le*)
show *?thesis* **using** *c101 c102* **by** *blast*
qed
qed
have *c11*: *i*≠*j* ⟹ *i* ≠ − ∞ ∧ *j* = ∞ ⟹
(∃ *ri*::*eint nellist*. *nnth ri* (*0*::*nat*) = *i* ∧ ¬ *nfinite ri* ∧
nridx (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧
((*upd* σ *n* (λ*x*::*int*. σ (*eint x*) (*eint x*) ⊨ *w*)) *a b* ⊨ *f*)) *ri*) ⟹
(∃ *ri*::*eint nellist*. *nnth ri* (*0*::*nat*) = *i* ∧ ¬ *nfinite ri* ∧
nridx (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧ (σ *a b* ⊨ *suform f n* (*w iand lfinite*))) *ri*)
proof −
assume *c11a0*: *i*≠*j*
assume *c11a1*: *i* ≠ − ∞ ∧ *j* = ∞
assume *c11a2*: (∃ *ri*::*eint nellist*. *nnth ri* (*0*::*nat*) = *i* ∧ ¬ *nfinite ri* ∧
nridx (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧
((*upd* σ *n* (λ*x*::*int*. σ (*eint x*) (*eint x*) ⊨ *w*)) *a b* ⊨ *f*)) *ri*)
show (∃ *ri*::*eint nellist*. *nnth ri* (*0*::*nat*) = *i* ∧ ¬ *nfinite ri* ∧
nridx (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧ (σ *a b* ⊨ *suform f n* (*w iand lfinite*))) *ri*)
proof −
obtain *ri* **where** *c111*: *nnth ri* (*0*::*nat*) = *i* ∧ ¬ *nfinite ri* ∧
nridx (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧
((*upd* σ *n* (λ*x*::*int*. σ (*eint x*) (*eint x*) ⊨ *w*)) *a b* ⊨ *f*)) *ri*
using *c11a2* **by** *blast*
have *c112*: *nridx* (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧ (σ *a b* ⊨ *suform f n* (*w iand lfinite*))) *ri*
using *chopstar-d.IH*[*of n w - - σ*] *c111 chopstar-d.prems*
by (*metis* (*no-types*, *lifting*) *bvars.simps*(*8*) *less-eint.simps*(*2*) *less-eint.simps*(*3*) *nridx-expand order-less-imp-le*)
show *?thesis* **using** *c111 c112* **by** *blast*
qed
qed
have *c12*: *i*≠*j* ∧ *i* ≠ − ∞ ∧ *j* = ∞ ⟹ *?thesis*
using *c1 c2* **apply** *simp*
using *c10 c11* **by** *argo*
have *c13*: *i* ≠*j* ⟹ *i* = − ∞ ∧ *j* = ∞ ⟹
(∃ (*li*::*int nellist*) *ri*::*int nellist*.
nnth li (*0*::*nat*) = *nnth ri* (*0*::*nat*) ∧ ¬ *nfinite li* ∧
nridx (λ(*a*::*int*) *b*::*int*. *b* < *a* ∧ (σ (*eint b*) (*eint a*) ⊨ *suform f n* (*w iand lfinite*))) *li* ∧
¬ *nfinite ri* ∧
nridx (λ(*a*::*int*) *b*::*int*. *a* < *b* ∧ (σ (*eint a*) (*eint b*) ⊨ *suform f n* (*w iand lfinite*))) *ri*) ⟹
(∃ (*li*::*int nellist*) *ri*::*int nellist*.
nnth li (*0*::*nat*) = *nnth ri* (*0*::*nat*) ∧
¬ *nfinite li* ∧
nridx (λ(*a*::*int*) *b*::*int*. *b* < *a* ∧
((*upd* σ *n* (λ*x*::*int*. σ (*eint x*) (*eint x*) ⊨ *w*)) (*eint b*) (*eint a*) ⊨ *f*)) *li* ∧
¬ *nfinite ri* ∧

```
      nridx (λ(a::int) b::int. a < b ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint a) (eint b) ⊨ f)) ri)
 proof −
  assume c13a0: i ≠j
  assume c13a1: i = − ∞ ∧ j = ∞
  assume c13a2: (∃(li::int nellist) ri::int nellist.
      nnth li (0::nat) = nnth ri (0::nat) ∧ ¬ nfinite li ∧
      nridx (λ(a::int) b::int. b < a ∧ (σ (eint b) (eint a) ⊨ suform f n (w iand lfinite))) li ∧
      ¬ nfinite ri ∧
      nridx (λ(a::int) b::int. a < b ∧ (σ (eint a) (eint b) ⊨ suform f n (w iand lfinite))) ri)
  show (∃(li::int nellist) ri::int nellist.
      nnth li (0::nat) = nnth ri (0::nat) ∧
      ¬ nfinite li ∧
      nridx (λ(a::int) b::int. b < a ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint b) (eint a) ⊨ f)) li ∧
      ¬ nfinite ri ∧
      nridx (λ(a::int) b::int. a < b ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint a) (eint b) ⊨ f)) ri)
  proof −
   obtain li ri where c131: nnth li (0::nat) = nnth ri (0::nat) ∧ ¬ nfinite li ∧
      nridx (λ(a::int) b::int. b < a ∧ (σ (eint b) (eint a) ⊨ suform f n (w iand lfinite))) li ∧
      ¬ nfinite ri ∧
      nridx (λ(a::int) b::int. a < b ∧ (σ (eint a) (eint b) ⊨ suform f n (w iand lfinite))) ri
   using c13a2 by blast
  have c132: nridx (λ(a::int) b::int. b < a ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint b) (eint a) ⊨ f)) li
   using  chopstar-d.IH[of n w - - σ] c131  chopstar-d.prems unfolding nridx-expand
     by simp  auto
  have c133: nridx (λ(a::int) b::int. a < b ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint a) (eint b) ⊨ f)) ri
    using  chopstar-d.IH[of n w - - σ] c131  chopstar-d.prems unfolding nridx-expand by auto
  show ?thesis using c131 c132 c133 by blast
 qed
qed
have c14: i ≠j ⟹ i = − ∞ ∧ j = ∞ ⟹
        (∃(li::int nellist) ri::int nellist.
      nnth li (0::nat) = nnth ri (0::nat) ∧
      ¬ nfinite li ∧
      nridx (λ(a::int) b::int. b < a ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint b) (eint a) ⊨ f)) li ∧
      ¬ nfinite ri ∧
      nridx (λ(a::int) b::int. a < b ∧
      ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint a) (eint b) ⊨ f)) ri ⟹
        (∃(li::int nellist) ri::int nellist.
      nnth li (0::nat) = nnth ri (0::nat) ∧ ¬ nfinite li ∧
      nridx (λ(a::int) b::int. b < a ∧ (σ (eint b) (eint a) ⊨ suform f n (w iand lfinite))) li ∧
      ¬ nfinite ri ∧
      nridx (λ(a::int) b::int. a < b ∧ (σ (eint a) (eint b) ⊨ suform f n (w iand lfinite))) ri)
  proof −
   assume c14a0: i ≠j
```

```
    assume c14a1: i = - ∞ ∧ j = ∞
    assume c14a2: (∃(li::int nellist) ri::int nellist.
        nnth li (0::nat) = nnth ri (0::nat) ∧
        ¬ nfinite li ∧
        nridx (λ(a::int) b::int. b < a ∧
        ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint b) (eint a) ⊨ f)) li ∧
        ¬ nfinite ri ∧
        nridx (λ(a::int) b::int. a < b ∧
        ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint a) (eint b) ⊨ f)) ri)
    show (∃(li::int nellist) ri::int nellist.
        nnth li (0::nat) = nnth ri (0::nat) ∧ ¬ nfinite li ∧
        nridx (λ(a::int) b::int. b < a ∧ (σ (eint b) (eint a) ⊨ suform f n (w iand lfinite))) li ∧
        ¬ nfinite ri ∧
        nridx (λ(a::int) b::int. a < b ∧ (σ (eint a) (eint b) ⊨ suform f n (w iand lfinite))) ri)
    proof −
     obtain li ri where c141: nnth li (0::nat) = nnth ri (0::nat) ∧
        ¬ nfinite li ∧
        nridx (λ(a::int) b::int. b < a ∧
        ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint b) (eint a) ⊨ f)) li ∧
        ¬ nfinite ri ∧
        nridx (λ(a::int) b::int. a < b ∧
        ((upd σ n (λx::int. σ (eint x) (eint x) ⊨ w)) (eint a) (eint b) ⊨ f)) ri
      using c14a2 by blast
     have c142: nridx (λ(a::int) b::int. b < a ∧ (σ (eint b) (eint a) ⊨ suform f n (w iand lfinite))) li
      using  chopstar-d.IH[of n w - - σ] c141  chopstar-d.prems  unfolding nridx-expand by auto
     have c143: nridx (λ(a::int) b::int. a < b ∧ (σ (eint a) (eint b) ⊨ suform f n (w iand lfinite))) ri
      using  chopstar-d.IH[of n w - - σ] c141  chopstar-d.prems  unfolding nridx-expand by auto
     show ?thesis using c141 c142 c143 by blast
    qed
   qed
  have c15: i ≠j ⟹ i = - ∞ ∧ j = ∞ ⟹ ?thesis
    using c13 c14 apply simp by argo
  show ?thesis
  using c12 c15 c3 c6 c9 by argo
 qed
next
case (exists-d x1 f)
then show ?case
    proof (cases n= x1)
    case True
    then show ?thesis using exists-d by simp
    next
    case False
    then show ?thesis
      proof −
       have e4: n ∉ bvars f
         using exists-d.prems(1) by auto
       have e5:  ((upd σ n (λc. σ (eint c) (eint c) ⊨ w)) i j ⊨ Ex x1. f)  =
         (∃ l.  (upd (upd σ n (λc. σ (eint c) (eint c) ⊨ w)) x1 l) i j ⊨ f)
         using exist-sigma-exist-value by auto
```

have *e6*: $(\bigwedge z.\ z \in fvars\ w \Longrightarrow z \notin bvars\ f)$
using *exists-d.prems(2)* **by** *auto*
have *e60*: $x1 \notin fvars\ w$
using *exists-d.prems(2)* **by** *auto*
have *e7*: $(\sigma\ i\ j \models Ex\ x1.\ suform\ f\ n\ (w\ iand\ lfinite)) =$
$(\exists\ l.\ (upd\ \sigma\ x1\ l)\ i\ j \models suform\ f\ n\ (w\ iand\ lfinite))$
using *exist-sigma-exist-value* **by** *auto*
have *e8*: $(\exists\ l.\ (upd\ \sigma\ x1\ l)\ i\ j \models suform\ f\ n\ (w\ iand\ lfinite)) \Longrightarrow$
$(\exists l.\ (upd\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)\ i\ j \models f)$
proof −
assume *ae0*: $(\exists\ l.\ (upd\ \sigma\ x1\ l)\ i\ j \models suform\ f\ n\ (w\ iand\ lfinite))$
show $(\exists l.\ (upd\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)\ i\ j \models f)$
proof −
obtain *l* **where** *e9*: $(upd\ \sigma\ x1\ l)\ i\ j \models suform\ f\ n\ (w\ iand\ lfinite)$
using *ae0* **by** *blast*
have *e10*: $((upd\ (upd\ \sigma\ x1\ l)\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w))\ i\ j \models f)$
using *exists-d.IH*[*of n w i j* $(upd\ \sigma\ x1\ l)$]
using *e4 e6 e9 exists-d.prems(3) exists-d.prems(4) exists-d.prems(5) exists-d.prems(6)* **by** *fastforce*

have *e11*: $(upd\ (upd\ \sigma\ x1\ l)\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w)) =$
$upd\ (upd\ \sigma\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w))\ x1\ l$
using *upd-swap*[*of x1 n* $\sigma\ l\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w)$]
using *False* **by** *blast*
have *e12*: $(\ upd\ (upd\ \sigma\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w))\ x1\ l) =$
$(upd\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)$
using *e60 not-fvar-upd* **by** *auto*
show *?thesis*
using *e10 e11 e12* **by** *auto*
qed
qed
have *e13*: $(\exists l.\ (upd\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)\ i\ j \models f) \Longrightarrow$
$(\exists\ l.\ (upd\ \sigma\ x1\ l)\ i\ j \models suform\ f\ n\ (w\ iand\ lfinite))$
proof −
assume *ae1*: $(\exists l.\ (upd\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)\ i\ j \models f)$
show $(\exists\ l.\ (upd\ \sigma\ x1\ l)\ i\ j \models suform\ f\ n\ (w\ iand\ lfinite))$
proof −
obtain *l* **where** *e14*: $(upd\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)\ i\ j \models f$
using *ae1* **by** *blast*
have *e15*: $(\ upd\ (upd\ \sigma\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w))\ x1\ l) =$
$(upd\ (upd\ \sigma\ n\ (\lambda c.\ \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)$
using *e60 not-fvar-upd* **by** *auto*
have *e16*: $(upd\ (upd\ \sigma\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w))\ x1\ l) =$
$(upd\ (upd\ \sigma\ x1\ l)\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w))$
using *False upd-swap* **by** *auto*
have *e17*: $((upd\ (upd\ \sigma\ x1\ l)\ n\ (\lambda c.\ (upd\ \sigma\ x1\ l)\ (eint\ c)\ (eint\ c) \models w))\ i\ j \models f)$
using *e14 e15 e16* **by** *auto*
have *e18*: $(upd\ \sigma\ x1\ l)\ i\ j \models suform\ f\ n\ (w\ iand\ lfinite)$
by (*simp add*: *e17 e4 e6 exists-d.IH exists-d.prems(3) exists-d.prems(4) exists-d.prems(5) exists-d.prems(6)*)
show *?thesis*

```
      using e18 by auto
     qed
    qed
    show ?thesis
    using e13 e5 e7 e8 by auto
   qed
   qed
qed

lemma state-nl-itl-point-interval:
fixes i :: int
assumes state-nl-itl w
        σ' i = σ i
shows  (σ' i i ⊨ w) ⟷(σ i i ⊨ w)
using assms
proof (induction w arbitrary: σ σ' i)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case by simp
next
case (iimp-d w1 w2)
then show ?case
using semantics-nl-itl.simps(3) state-nl-itl.simps(3) by blast
next
case (next-d w)
then show ?case by simp
next
case (chop-d w1 w2)
then show ?case by simp
next
case (ldiamond-d w)
then show ?case by simp
next
case (rdiamond-d w)
then show ?case by simp
next
case (chopstar-d w)
then show ?case by simp
next
case (exists-d x1 w)
then show ?case
 proof −
  have 1: (σ (eint i) (eint i) ⊨ Ex x1. w) = (∃ l. ((upd σ x1 l) (eint i) (eint i) ⊨ w)) (is  - = ?lhs)
   by simp
  have 2: (σ' (eint i) (eint i) ⊨ Ex x1. w) = (∃ l. ((upd σ' x1 l) (eint i) (eint i) ⊨ w)) (is  - = ?rhs)
    by simp
  have 3: ?lhs ⟹ ?rhs
   proof −
```

```
    assume a: ?lhs
    show ?rhs
    proof −
     obtain l where 4: ((upd σ x1 l) (eint i) (eint i) ⊨ w)
       using a by blast
     have 5: (upd σ x1 l) i = (upd σ′ x1 l) i
       using exists-d.prems(2) upd-def by presburger
      show ?thesis
      using 4 5 exists-d.IH exists-d.prems(1) state-nl-itl.simps(9) by blast
    qed
   qed
  have 4: ?rhs ⟹ ?lhs
   proof −
    assume b: ?rhs
    show ?lhs
    proof −
     obtain l where 6: ((upd σ′ x1 l) (eint i) (eint i) ⊨ w)
       using b by blast
     have 7: (upd σ′ x1 l) i = (upd σ x1 l) i
      using exists-d.prems(2) upd-def by presburger
     show ?thesis
     using 6 7 exists-d.IH exists-d.prems(1) state-nl-itl.simps(9) by blast
   qed
  qed
  show ?thesis
  using 1 2 3 4 by blast
 qed
qed


lemma exists-elim-state-nl-itl-help:
assumes
        i ≠ ∞
      state-nl-itl w
        n ∉ bvars w
       i≠ −∞
shows  (∃ l. ((upd σ n l) i i ⊨ w)) ⟷
              (((upd σ n (λx::int. True)) i i ⊨ w) ∨  ((upd σ n (λx::int. False)) i i ⊨ w))
using assms
proof (induction w arbitrary: σ n i)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case using upd-def by auto
next
case (iimp-d w1 w2)
then show ?case
 proof −
  have 1: state-nl-itl  w1
```

using *iimp-d.prems(2)* **by** *auto*
have *2*: *state-nl-itl w2*
using *iimp-d.prems(2)* **by** *auto*
have *3*: $n \notin bvars\ w1$
using *iimp-d.prems(3)* **by** *simp*
have *4*: $n \notin bvars\ w2$
using *iimp-d.prems(3)* **by** *simp*
have *5*: $(\exists\, l.\ (upd\ \sigma\ n\ l)\ i\ i \models w1\ iimp\ w2\ ) =$
$(\exists\, l.\ ((upd\ \sigma\ n\ l)\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ l)\ i\ i \models w2))$
by *simp*
have *6*: $((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w1\ iimp\ w2\ ) =$
$(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2))$
by *simp*
have *7*: $((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w1\ iimp\ w2\ ) =$
$(((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2))$
by *simp*
have *8*: $(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2)) \Longrightarrow$
$(\exists\, l.\ ((upd\ \sigma\ n\ l)\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ l)\ i\ i \models w2))$
by *blast*
have *9*: $(((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2)) \Longrightarrow$
$(\exists\, l.\ ((upd\ \sigma\ n\ l)\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ l)\ i\ i \models w2))$
by *blast*
have *10*: $(\exists\, l.\ ((upd\ \sigma\ n\ l)\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ l)\ i\ i \models w2)) \Longrightarrow$
$(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2)) \vee$
$(((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2))$
proof −
assume *a1*: $(\exists\, l.\ ((upd\ \sigma\ n\ l)\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ l)\ i\ i \models w2))$
show $(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2)) \vee$
$(((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2))$
proof −
obtain *l* **where** *11*: $((upd\ \sigma\ n\ l)\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ l)\ i\ i \models w2)$
using *a1* **by** *blast*
have *111*: $\exists\ i1\ .\ eint\ i1 = i$
by (*metis MInfty-eq-minfinity PInfty-eq-infinity eint.exhaust iimp-d.prems(1,4)*)
obtain *i1* **where** *112*: *eint i1* = *i*
using *111* **by** *blast*
have *12* : $((upd\ \sigma\ n\ l)\ i\ i \models w1) \Longrightarrow$
$(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w1) \vee ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w1))$
using *1 3 iimp-d.IH(1) iimp-d.prems(1)*
using *iimp-d.prems(4)* **by** *blast*
have *13*: $((upd\ \sigma\ n\ l)\ i\ i \models w2) \Longrightarrow$
$(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2) \vee ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2))$
using *2 4 iimp-d.IH(2) iimp-d.prems(1)* **using** *iimp-d.prems(4)* **by** *blast*
have *14*: $(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2) \vee ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2)) \Longrightarrow$
$(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2) \vee ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2))$
by (*simp add*: *11 13*)
have *141*: $(upd\ \sigma\ n\ (\lambda x.\ True))\ i1 \neq (upd\ \sigma\ n\ (\lambda x.\ False))\ i1$
unfolding *upd-def* **by** *auto*
have *15*: $(((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ True))\ i\ i \models w2)) \vee$
$(((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w1) \longrightarrow ((upd\ \sigma\ n\ (\lambda x.\ False))\ i\ i \models w2))$

using *1 11 12 13 112 141 iimp-d.prems state-nl-itl-point-interval*[*of w1 - i1*]
by (*metis upd-def*)
show *?thesis*
using *15* **by** *blast*
qed
qed
show *?thesis*
using *10 5 6 7* **by** *blast*
qed
next
case (*next-d w*)
then show *?case* **by** *simp*
next
case (*chop-d w1 w2*)
then show *?case* **by** *simp*
next
case (*ldiamond-d w*)
then show *?case* **by** *simp*
next
case (*rdiamond-d w*)
then show *?case* **by** *simp*
next
case (*chopstar-d w*)
then show *?case* **by** *simp*
next
case (*exists-d x1 w*)
then show *?case* **apply** *simp*
by (*metis upd-swap*)
qed

lemma *suform-lfinite-export*:
⊢ *suform* (*f iand lfinite*) *n g ieqv* (*lfinite iand* (*suform f n g*))
unfolding *valid-def* **unfolding** *iand-d-def lfinite-d-def inot-d-def ior-d-def itrue-d-def empty-d-def more-d-def*
apply *simp*
by *meson*

lemma *subst-suform-state*:
assumes $n \notin$ *bvars f*
$\bigwedge z\ .\ z \in$ *fvars w* $\Longrightarrow z \notin$ *bvars f*
state-nl-itl w
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
state-nl-itl f
shows (σ *i j* $\models$ *suform* (*f iand lfinite*) *n* (*w*)) = (σ *i j* $\models$ *suform* (*f iand lfinite*) *n* (*w iand lfinite*))
using *assms*
proof (*induction f arbitrary*: σ)
case *false-d*

then show *?case*
using *itl-ieq suform.simps(1) suform-lfinite-export* **by** *presburger*
next
case (*atom-d x*)
then show *?case*
using *suform-lfinite-export* **by** *auto*
next
case (*iimp-d f1 f2*)
then show *?case*
proof −
have *i1*: (*σ i j* ⊨ *suform* ((*inot f1*) *iand lfinite*) *n w*) = (*σ i j* ⊨ *suform* ((*inot f1*) *iand lfinite*) *n* (*w iand lfinite*))
using *iimp-d inot-d-def suform-lfinite-export* **by** *force*
have *i2*: (*σ i j* ⊨ *suform* (*f2 iand lfinite*) *n w*) = (*σ i j* ⊨ *suform* (*f2 iand lfinite*) *n* (*w iand lfinite*))
using *iimp-d* **by** *force*
show *?thesis*
using *i1 i2 iimp-d.prems(4) iimp-d.prems(5) iimp-d.prems(6) inot-d-def suform-lfinite-export* **by** *auto*
qed
next
case (*next-d f*)
then show *?case* **by** *simp*
next
case (*chop-d f1 f2*)
then show *?case* **by** *simp*
next
case (*ldiamond-d f*)
then show *?case* **by** *simp*
next
case (*rdiamond-d f*)
then show *?case* **by** *simp*
next
case (*chopstar-d f*)
then show *?case* **by** *simp*
next
case (*exists-d x1 f*)
then show *?case*
proof −
have *e1*: (*σ i j* ⊨ *suform* ((*Ex x1. f*) *iand lfinite*) *n w*) =
(*σ i j* ⊨ *suform* ((*Ex x1. f iand lfinite*)) *n w*)
using *exists-d.prems(4) exists-d.prems(5) exists-d.prems(6) suform-lfinite-export* **by** *auto*
have *e2*: (*σ i j* ⊨ *suform* ((*Ex x1. f*) *iand lfinite*) *n* (*w iand lfinite*)) =
(*σ i j* ⊨ *suform* ((*Ex x1. f iand lfinite*)) *n* (*w iand lfinite*))
using *exists-d.prems(4) exists-d.prems(5) exists-d.prems(6) suform-lfinite-export* **by** *force*
have *e3*: (*σ i j* ⊨ *suform* (*f iand lfinite*) *n w*) = (*σ i j* ⊨ *suform* (*f iand lfinite*) *n* (*w iand lfinite*))
using *exists-d* **by** *auto*
show *?thesis*
using *e1 e2 exists-d.IH exists-d.prems(1) exists-d.prems(2) exists-d.prems(3) exists-d.prems(4) exists-d.prems(5) exists-d.prems(6) exists-d.prems(7)* **by** *force*
qed
qed

lemma *subst-suform-state-1*:
assumes $n \notin bvars\ f$
$\bigwedge z\ .\ z \in fvars\ w \Longrightarrow z \notin bvars\ f$
state-nl-itl w
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
state-nl-itl f
shows $(\sigma\ i\ j \models\ suform\ (f\ iand\ lfinite)\ n\ (w)) = ((upd\ \sigma\ n\ (\lambda\ c.\ (\sigma\ c\ c \models w\ )\ )\ )\ \ i\ j \models f\ iand\ lfinite)$
proof −
have *1*: $(n::nat) \notin bvars\ ((f::nl\text{-}itl)\ iand\ lfinite)$
unfolding *iand-d-def inot-d-def lfinite-d-def itrue-d-def empty-d-def more-d-def*
by (*simp add*: *assms*(*1*) *inot-d-def ior-d-def*)
have *2*: $(\bigwedge z::nat.\ z \in fvars\ (w::nl\text{-}itl) \Longrightarrow z \notin bvars\ (f\ iand\ lfinite))$
unfolding *iand-d-def inot-d-def lfinite-d-def itrue-d-def empty-d-def more-d-def ior-d-def*
by (*simp add*: *assms*(*2*))
have *3*: $(\sigma\ i\ j \models\ suform\ (f\ iand\ lfinite)\ n\ (w\ iand\ lfinite)) = ((upd\ \sigma\ n\ (\lambda\ c.\ (\sigma\ c\ c \models w\ )\ )\ )\ \ i\ j \models f\ iand\ lfinite)$
using *subst-suform*[*of n f iand lfinite w i j σ*] *assms*
using *1 2* **by** *blast*
have *4*: $(\sigma\ i\ j \models\ suform\ (f\ iand\ lfinite)\ n\ (w\ iand\ lfinite)) =$
$(\sigma\ i\ j \models\ suform\ (f\ iand\ lfinite)\ n\ (w\ ))$
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*6*) *assms*(*7*) *subst-suform-state*)
show *?thesis*
using *3 4* **by** *blast*
qed

lemma *rename-nl-pitl*:
assumes $v1 \notin bvars\ f$
$v2 \notin bvars\ f$
$v1 \neq v2$
$v2 \notin fvars\ f$
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models Ex\ v1.\ f) = (\sigma\ i\ j \models Ex\ v2.\ suform\ f\ v1\ (\$v2\ iand\ lfinite))$
proof −
have *1*: $(\sigma\ i\ j \models Ex\ v1.\ f) = (\exists\ l.\ ((upd\ \sigma\ v1\ l)\ i\ j \models f))$
using *exist-sigma-exist-value* **by** *auto*
have *2*: $(\sigma\ i\ j \models Ex\ v2.\ suform\ f\ v1\ (\$v2\ iand\ lfinite)) =$
$(\exists\ l.\ ((upd\ \sigma\ v2\ l)\ i\ j \models suform\ f\ v1\ (\$v2\ iand\ lfinite))\ )$
using *exist-sigma-exist-value* **by** *auto*
have *3*: $(\exists\ l.\ ((upd\ \sigma\ v2\ l)\ i\ j \models suform\ f\ v1\ (\$v2\ iand\ lfinite))\ ) \Longrightarrow$
$(\exists\ l.\ ((upd\ \sigma\ v1\ l)\ i\ j \models f))$
proof −

```
  assume a0: (∃ l. ((upd σ v2 l) i j ⊨ suform f v1 ($v2 iand lfinite) ))
  show (∃ l. ((upd σ v1 l) i j ⊨ f))
  proof −
   obtain l where 4: ((upd σ v2 l) i j ⊨ suform f v1 ($v2 iand lfinite) )
    using a0 by blast
    have 5: ((upd σ v2 l) i j ⊨ suform f v1 ($v2 iand lfinite) ) =
              ((upd (upd σ v2 l) v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) i j ⊨ f)
     using subst-suform[of v1 f $v2 i j  (upd σ v2 l)  ]
    by (simp add: assms(1) assms(2) assms(5) assms(6) assms(7))
   have 6: (upd (upd σ v2 l) v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) =
            upd (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) v2 l
     using upd-swap[of v2 v1 σ l (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)]
   using assms(3) by blast
   have 8: ((upd (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) v2 l) i j ⊨ f) =
           ( (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) i j ⊨ f)
     using not-fvar-upd[of v2 f ] using assms(4) by blast
   show ?thesis using 4 5 6 8 by auto
  qed
 qed
 have 9: (∃ l. ((upd σ v1 l) i j ⊨ f)) ⟹
         (∃ l. ((upd σ v2 l) i j ⊨ suform f v1 ($v2 iand lfinite) ))
 proof −
  assume a1: (∃ l. ((upd σ v1 l) i j ⊨ f))
  show (∃ l. ((upd σ v2 l) i j ⊨ suform f v1 ($v2 iand lfinite) ))
  proof −
   obtain l where 10: ((upd σ v1 l) i j ⊨ f)
    using a1 by blast
   have 11: ( (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) i j ⊨ f) =
            ((upd (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) v2 l) i j ⊨ f)

    using not-fvar-upd[of v2 f ] using assms(4) by blast
   have 12: ⋀k. (upd σ v1 l) k = (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) k
     using upd-def by fastforce
   have 13: (upd σ v1 l) = (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2))
    using 12 by blast
   have 14: ((upd σ v2 l) i j ⊨ suform f v1 ($v2 iand lfinite) ) =
            ((upd (upd σ v2 l) v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) i j ⊨ f)
     using subst-suform[of v1 f $v2 i j  (upd σ v2 l) ]
      by (simp add: assms(1) assms(2) assms(5) assms(6) assms(7))
   have 15: (upd (upd σ v2 l) v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) =
            (upd (upd σ v1 (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)) v2 l)
    using upd-swap[of v2 v1 σ l (λx::int. (upd σ v2 l) (eint x) (eint x) ⊨ $v2)]
    using assms(3) by blast
   show ?thesis
   using 10 11 13 14 15 by auto
  qed
 qed
 show ?thesis
 using 1 2 3 9 by blast
qed
```

lemma *atom-lfinite-absorb*:
$\vdash$ (\$*n*) *ieqv* \$*n iand lfinite*
unfolding *valid-def* **apply** *simp* **by** *meson*

lemma *suform-ieqv-sem*:
assumes $\bigwedge \sigma\ i\ j\ .\ i \leq j \wedge\ \ i \neq \infty \wedge j \neq -\infty \Longrightarrow (\sigma\ i\ j \models\ f) = (\sigma\ i\ j \models\ g)$
$i \leq j$
$i \neq \infty$
$j \neq -\infty$
shows $(\sigma\ i\ j \models (suform\ h\ n\ f)) = (\sigma\ i\ j \models\ (suform\ h\ n\ g))$
using *assms*
proof (*induction h arbitrary*: $\sigma\ i\ j$)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case* **by** *simp*
next
case (*iimp-d h1 h2*)
then show *?case*
proof −
have *i1*: $(\sigma\ i\ j \models\ suform\ (inot\ h1)\ n\ f\ ieqv\ suform\ (inot\ h1)\ n\ g)$
using *assms iimp-d.IH*(*1*) **unfolding** *inot-d-def* **using** *iimp-d.prems* **by** *auto*
have *i2*: $(\sigma\ i\ j \models\ suform\ (h2)\ n\ f\ ieqv\ suform\ (h2)\ n\ g)$
using *assms iimp-d.IH*(*1*) *iimp-d.prems* **using** *iimp-d.IH*(*2*) *ieqv-defs* **by** *presburger*
show *?thesis* **using** *i1 i2 inot-d-def* **by** *auto*
qed
next
case (*next-d h*)
then show *?case*
proof −
have *n1*: $(\sigma\ i\ j \models suform\ (next\ h)\ n\ f) = (i \neq -\infty \wedge i < j \wedge (\sigma\ (i + eint\ (1))\ j \models suform\ h\ n\ f))$
by *simp*
have *n2*: $(\sigma\ i\ j \models suform\ (next\ h)\ n\ g) = (i \neq -\infty \wedge i < j \wedge (\sigma\ (i + eint\ (1))\ j \models suform\ h\ n\ g))$
by *simp*
have *n3*: $i \neq -\infty \wedge i < j \Longrightarrow (\sigma\ (i + eint\ (1))\ j \models suform\ h\ n\ f) = (\sigma\ (i + eint\ (1))\ j \models suform\ h\ n\ g)$
using *next-d.IH*[*of* $(i + eint\ (1))\ j\ \sigma$] *next-d.prems* **apply** *auto*
apply (*metis eint-less-le-alt int-of-eint.cases plus-eint.simps*(*1*))
by (*metis eint-less-le-alt int-of-eint.cases plus-eint.simps*(*1*))
show *?thesis*
using *n1 n2 n3* **by** *blast*
qed
next
case (*ldiamond-d h*)

then show *?case* **apply** *auto*
apply (*metis eint-less-eq(1) less-eq-eint-def linorder-not-le*)
by (*metis eint-less-eq(1) linorder-not-le order-less-le*)
next
case (*rdiamond-d h*)
then show *?case* **apply** *auto*
apply (*metis less-eint.simps(3) linorder-not-le not-less-iff-gr-or-eq*)
by (*metis eint-less-eq(2) linorder-not-less order-less-le*)
next
case (*chopstar-d h*)
then show *?case*
proof −
have *c1*: $i = j \Longrightarrow$ *?thesis*
using *semantics-nl-itl.simps(8) suform.simps(8)* **by** *presburger*
have *c10*: $i{\neq}j \wedge i \neq -\infty \wedge j \neq \infty \Longrightarrow$ *?thesis*
proof −
assume *c10a*: $i{\neq}j \wedge i \neq -\infty \wedge j \neq \infty$
show $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f) = (\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
proof −
have *c11*: $\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f \Longrightarrow \sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g$
proof −
assume *c11a*: $\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f$
show $\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g$
proof −
have *c12*: $\exists\, lrf.\ nnth\ lrf\ (0{::}nat) = i \wedge nfinite\ lrf \wedge nlast\ lrf = j \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ f))\ lrf$
using *c10a c11a* **by** *simp*
obtain *lrf* **where** *c13*: $nnth\ lrf\ (0{::}nat) = i \wedge nfinite\ lrf \wedge nlast\ lrf = j \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ f))\ lrf$
using *c12* **by** *blast*
have *c14*: $nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ g))\ lrf$
using *chopstar-d.IH c13 nridx-expand*
by (*metis* (*no-types*, *lifting*) *chopstar-d.prems(1) eint-less-eq(1) less-eint.simps(3) linorder-not-le order-less-le*)
have *c15*: $\exists\, lrf.\ nnth\ lrf\ (0{::}nat) = i \wedge nfinite\ lrf \wedge nlast\ lrf = j \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ g))\ lrf$
using *c14 c13* **by** *blast*
show *?thesis* **using** *c15 c10a* **by** *simp*
qed
qed
have *c16*: $\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g \Longrightarrow \sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f$
proof −
assume *c16a*: $\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g$
show $\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f$
proof −
have *c17*: $\exists\, lrf.\ nnth\ lrf\ (0{::}nat) = i \wedge nfinite\ lrf \wedge nlast\ lrf = j \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ g))\ lrf$
using *c10a c16a* **by** *simp*
obtain *lrf* **where** *c18*: $nnth\ lrf\ (0{::}nat) = i \wedge nfinite\ lrf \wedge nlast\ lrf = j \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ g))\ lrf$

using *c17* **by** *blast*
have *c19*: *nridx* (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧ (σ *a b* ⊨ *suform h n f*)) *lrf*
using *chopstar-d.IH c18 nridx-expand*
by (*metis* (*no-types*, *lifting*) *chopstar-d.prems*(*1*) *eint-less-eq*(*1*) *less-eint.simps*(*3*) *linorder-not-le order-less-le*)
have *c19a*: ∃ *lrf*. *nnth lrf* (*0*::*nat*) = *i* ∧ *nfinite lrf* ∧ *nlast lrf* = *j* ∧ *nridx* (λ(*a*::*eint*) *b*::*eint*. *a* < *b* ∧ (σ *a b* ⊨ *suform h n f*)) *lrf*
using *c18 c19* **by** *blast*
show *?thesis* **using** *c19a c10a* **by** *simp*
qed
qed
show *?thesis*
using *c11 c16* **by** *blast*
qed
qed
have *c20*: *i*≠*j* ∧ *i* = −∞ ∧ *j* ≠ ∞ ⟹ *?thesis*
proof −
assume *c20a*: *i*≠*j* ∧ *i* = −∞ ∧ *j* ≠ ∞
show (σ *i j* ⊨ *suform* (*chopstar h*) *n f*) = (σ *i j* ⊨ *suform* (*chopstar h*) *n g*)
proof −
have *c21*: σ *i j* ⊨ *suform* (*chopstar h*) *n f* ⟹ σ *i j* ⊨ *suform* (*chopstar h*) *n g*
proof −
assume *c21a*: σ *i j* ⊨ *suform* (*chopstar h*) *n f*
show σ *i j* ⊨ *suform* (*chopstar h*) *n g*
proof −
have *c22*: (∃ *li*::*eint nellist*. *nnth li* (*0*::*nat*) = *j* ∧ ¬ *nfinite li* ∧ *nridx* (λ(*a*::*eint*) *b*::*eint*. *b* < *a* ∧ (σ *b a* ⊨ *suform h n f*)) *li*)
using *c20a c21a* **by** *fastforce*
obtain *li* **where** *c23*: *nnth li* (*0*::*nat*) = *j* ∧ ¬ *nfinite li* ∧ *nridx* (λ(*a*::*eint*) *b*::*eint*. *b* < *a* ∧ (σ *b a* ⊨ *suform h n f*)) *li*
using *c22* **by** *blast*
have *c24*: *nridx* (λ(*a*::*eint*) *b*::*eint*. *b* < *a* ∧ (σ *b a* ⊨ *suform h n g*)) *li*
by (*metis* (*no-types*, *lifting*) *c23 chopstar-d.IH chopstar-d.prems*(*1*) *eint-less-eq*(*1*) *less-eint.simps*(*3*) *linorder-not-le nridx-expand order-less-le*)
show *?thesis* **using** *c20a c23 c24* **by** *auto*
qed
qed
have *c25*: σ *i j* ⊨ *suform* (*chopstar h*) *n g* ⟹ σ *i j* ⊨ *suform* (*chopstar h*) *n f*
proof −
assume *c25a*: σ *i j* ⊨ *suform* (*chopstar h*) *n g*
show σ *i j* ⊨ *suform* (*chopstar h*) *n f*
proof −
have *c26*: (∃ *li*::*eint nellist*. *nnth li* (*0*::*nat*) = *j* ∧ ¬ *nfinite li* ∧ *nridx* (λ(*a*::*eint*) *b*::*eint*. *b* < *a* ∧ (σ *b a* ⊨ *suform h n g*)) *li*)
using *c25a c20a* **by** *fastforce*
obtain *li* **where** *c27*: *nnth li* (*0*::*nat*) = *j* ∧ ¬ *nfinite li* ∧ *nridx* (λ(*a*::*eint*) *b*::*eint*. *b* < *a* ∧ (σ *b a* ⊨ *suform h n g*)) *li*
using *c26* **by** *blast*
have *c28*: *nridx* (λ(*a*::*eint*) *b*::*eint*. *b* < *a* ∧ (σ *b a* ⊨ *suform h n f*)) *li*
by (*metis* (*no-types*, *lifting*) *c27 chopstar-d.IH chopstar-d.prems*(*1*) *eint-less-eq*(*1*) *less-eint.simps*(*3*)

linorder-not-le nridx-expand order-less-le)
show *?thesis* **using** *c20a c27 c28* **by** *auto*
qed
qed
show *?thesis*
using *c21 c25* **by** *blast*
qed
qed
have *c30*: $i{\neq}j \wedge i \neq -\infty \wedge j = \infty \Longrightarrow$ *?thesis*
proof −
assume *c30a*: $i{\neq}j \wedge i \neq -\infty \wedge j = \infty$
show $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f) = (\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
proof −
have *c31*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f) \Longrightarrow (\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
proof −
assume *c31a*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f)$
show $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
proof −
have *c32*: $(\exists\, ri{::}eint\ nellist.\ nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ f))\ ri)$
using *c30a c31a* **by** *auto*
obtain *ri* **where** *c33*: $nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ f))\ ri$
using *c32* **by** *blast*
have *c34*: $nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ g))\ ri$
by (*metis* (*no-types*, *lifting*) *c33 chopstar-d.IH chopstar-d.prems*(*1*) *eint-less-eq*(*1*) *less-eint.simps*(*3*) *linorder-not-le nridx-expand order-less-le*)
show *?thesis* **using** *c33 c34 c30a* **by** *auto*
qed
qed
have *c35*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g) \Longrightarrow (\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f)$
proof −
assume *c35a*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
show $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f)$
proof −
have *c36*: $(\exists\, ri{::}eint\ nellist.\ nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ g))\ ri)$
using *c30a c35a* **by** *auto*
obtain *ri* **where** *c37*: $nnth\ ri\ (0{::}nat) = i \wedge \neg\ nfinite\ ri \wedge nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ g))\ ri$
using *c36* **by** *blast*
have *c38*: $nridx\ (\lambda(a{::}eint)\ b{::}eint.\ a < b \wedge (\sigma\ a\ b \models suform\ h\ n\ f))\ ri$
by (*metis* (*no-types*, *lifting*) *c37 chopstar-d.IH chopstar-d.prems*(*1*) *eint-less-PInfty less-eint.simps*(*3*) *not-less-iff-gr-or-eq nridx-expand order-less-le*)
show *?thesis* **using** *c37 c38 c30a* **by** *auto*
qed
qed
show *?thesis* **using** *c35 c31* **by** *blast*
qed
qed

have *c40*: $i{\neq}j \wedge i = -\infty \wedge j = \infty \Longrightarrow$ *?thesis*
proof −
assume *c40a*: $i{\neq}j \wedge i = -\infty \wedge j = \infty$
show $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f) = (\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
proof −
have *c41*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f) \Longrightarrow (\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
proof −
assume *c41a*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f)$
show $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
proof −
have *c42*: $\exists (li{::}int\ nellist)\ ri{::}int\ nellist.$
$nnth\ li\ (0{::}nat) = nnth\ ri\ (0{::}nat) \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ h\ n\ f))\ li \wedge$
$\neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ h\ n\ f))\ ri$
using *c40a c41a* **by** *simp*
obtain *li* **and** *ri* **where** *c43*: $nnth\ li\ (0{::}nat) = nnth\ ri\ (0{::}nat) \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ h\ n\ f))\ li \wedge$
$\neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ h\ n\ f))\ ri$
using *c42* **by** *blast*
have *c44a*: $nridx\ (\lambda(a{::}int)\ b{::}int.\ b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ h\ n\ g))\ li$
by (*metis* (*no-types*, *lifting*) *PInfty-neq-eint*(*2*) *c43 chopstar-d.IH chopstar-d.prems*(*1*) *less-eint.simps*(*1*) *less-eint.simps*(*3*) *less-eq-eint-def nridx-expand*)
have *c44b*: $nridx\ (\lambda(a{::}int)\ b{::}int.\ a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ h\ n\ g))\ ri$
by (*metis* (*no-types*, *lifting*) *PInfty-neq-eint*(*2*) *c43 chopstar-d.IH chopstar-d.prems*(*1*) *less-eint.simps*(*1*) *less-eint.simps*(*3*) *less-eq-eint-def nridx-expand*)
show *?thesis* **using** *c40a c44a c44b c43* **by** *auto*
qed
qed
have *c45*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g) \Longrightarrow (\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f)$
proof −
assume *c45a*: $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ g)$
show $(\sigma\ i\ j \models suform\ (chopstar\ h\ )\ n\ f)$
proof −
have *c46*: $\exists (li{::}int\ nellist)\ ri{::}int\ nellist.$
$nnth\ li\ (0{::}nat) = nnth\ ri\ (0{::}nat) \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ h\ n\ g))\ li \wedge$
$\neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ h\ n\ g))\ ri$
using *c40a c45a* **by** *simp*
obtain *li* **and** *ri* **where** *c47*: $nnth\ li\ (0{::}nat) = nnth\ ri\ (0{::}nat) \wedge \neg\ nfinite\ li \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ h\ n\ g))\ li \wedge$
$\neg\ nfinite\ ri \wedge$
$nridx\ (\lambda(a{::}int)\ b{::}int.\ a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ h\ n\ g))\ ri$
using *c46* **by** *blast*
have *c48a*: $nridx\ (\lambda(a{::}int)\ b{::}int.\ b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ h\ n\ f))\ li$
by (*metis* (*no-types*, *lifting*) *PInfty-neq-eint*(*2*) *c47 chopstar-d.IH chopstar-d.prems*(*1*) *less-eint.simps*(*1*) *less-eint.simps*(*3*) *less-eq-eint-def nridx-expand*)
have *c48b*: $nridx\ (\lambda(a{::}int)\ b{::}int.\ a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ h\ n\ f))\ ri$

by (*metis* (*no-types*, *lifting*) *PInfty-neq-eint*(*2*) *c47 chopstar-d.IH chopstar-d.prems*(*1*) *less-eint.simps*(*1*) *less-eint.simps*(*3*) *less-eq-eint-def nridx-expand*)
show *?thesis* **using** *c40a c48a c48b c47* **by** *auto*
qed
qed
show *?thesis* **using** *c45 c41* **by** *blast*
qed
qed
show *?thesis*
using *c1 c10 c20 c30 c40* **by** *argo*
qed
next
case (*exists-d x1 h*)
then show *?case*
by *auto*
qed

lemma *suform-ieqv*:
assumes $\vdash$ *f ieqv g*
shows $\vdash$ (*suform h n f*) *ieqv* (*suform h n g*)
using *assms* **unfolding** *valid-def* **using** *suform-ieqv-sem*
by *fastforce*

lemma *rename-nl-pitl-1*:
assumes *v1* $\notin$ *bvars f*
v2 $\notin$ *bvars f*
v1 $\neq$ *v2*
v2 $\notin$ *fvars f*
i $\leq$ *j*
i $\neq \infty$
j $\neq -\infty$
shows (σ *i j* $\models$ *Ex v1. f*) = (σ *i j*$\models$ *Ex v2. suform f v1* ($v2))
proof −
have *1*: (σ *i j* $\models$ *Ex v1. f*) = (σ *i j*$\models$ *Ex v2. suform f v1* ($v2 *iand lfinite*))
using *assms rename-nl-pitl* **by** *blast*
have *2*: $\vdash$ $v2 *ieqv* $v2 *iand lfinite*
using *atom-lfinite-absorb* **by** *blast*
have *3*: $\vdash$ *suform f v1* ($v2 *iand lfinite*) *ieqv suform f v1* ($v2)
using *2 suform-ieqv-sem* **by** *fastforce*
have *4*: (σ *i j*$\models$ *Ex v2. suform f v1* ($v2 *iand lfinite*)) =
(σ *i j*$\models$ *Ex v2. suform f v1* ($v2))
using *3 assms*(*5*) *assms*(*6*) *assms*(*7*) **by** *auto*
show *?thesis*
using *1 4* **by** *blast*
qed

lemma *ExistSubst*:
assumes *n* $\notin$ *bvars f*

$\bigwedge z\ .\ z \in fvars\ w \Longrightarrow z \notin bvars\ f$
$state\text{-}nl\text{-}itl\ w$
shows $\vdash suform\ f\ n\ (w\ iand\ lfinite)\ iimp\ (Ex\ n.\ f)$
using *assms* **unfolding** *valid-def*
using *similar-upd*[*of - n*] *subst-suform*[*of n f w*]
using *semantics-nl-itl.simps*(*3*) *semantics-nl-itl.simps*(*9*) **by** *blast*

lemma *ForallSubst*:
assumes $n \notin bvars\ f$
$\bigwedge z\ .\ z \in fvars\ w \Longrightarrow z \notin bvars\ f$
$state\text{-}nl\text{-}itl\ w$
shows $\vdash (Fa\ n.\ f)\ iimp\ suform\ f\ n\ (w\ iand\ lfinite)$
using *assms*
using *forall-d-def similar-upd subst-suform* **by** *auto*

lemma *ExistsImp*:
assumes $v1 \notin fvars\ f1$
shows $\vdash (Ex\ v1.\ (f1\ iimp\ f2))\ ieqv\ (f1\ iimp\ (Ex\ v1.\ f2))$
unfolding *valid-def* **apply** *simp*
by (*meson assms exist-sigma-exist-value not-fvar-upd similar-commute similar-refl*)

lemma *ExistsSChopRight*:
assumes $v1 \notin fvars\ f2$
shows $\vdash (Ex\ v1.\ f1\ schop\ f2)\ iimp\ (Ex\ v1.\ f1)\ schop\ f2$
using *assms*
unfolding *valid-def* **apply** *auto*
using *not-fvar-upd* **by** *blast*

lemma *ExistsSChopLeft*:
assumes $v1 \notin fvars\ f1$
shows $\vdash (Ex\ v1.\ f1\ schop\ f2)\ iimp\ f1\ schop\ (Ex\ v1.\ f2)$
using *assms*
unfolding *valid-def* **apply** *auto*
using *not-fvar-upd* **by** *blast*

lemma *ForallGen*:
assumes $\vdash f$
shows $\vdash Fa\ v.\ f$
using *assms*
by (*simp add*: *forall-d-def*)

lemma *ExistsI*:
assumes $i \leq j$
$i \neq \infty$
$j \neq -\infty$
$(\sigma\ i\ j \models f)$
shows $(\sigma\ i\ j \models Ex\ v.\ f)$
using *assms* **apply** *simp*
using *exist-sigma-exist-value similar-refl* **by** *blast*

lemma *init-state-nl-itl*:
assumes *state-nl-itl w*
shows $\vdash$ *init w ieqv lfinite iand w*
using *assms*
unfolding *valid-def*
by (*simp add*: *init-defs state-nl-itl-defs*)

lemma *exists-elim-state-nl-itl*:
assumes *state-nl-itl w*
$n \notin$ *bvars w*
shows $\vdash$ (*Ex n. w iand lfinite*) *ieqv*
(*suform* (*w iand lfinite*) *n* (*itrue iand lfinite*)) *ior*
(*suform* (*w iand lfinite*) *n* (*ifalse iand lfinite*))
proof −
have *1*: *state-nl-itl itrue*
by (*simp add*: *inot-d-def itrue-d-def*)
have *2*: ($\bigwedge z.\ z \in$ *fvars itrue* $\Longrightarrow z \notin$ *bvars w*)
by (*simp add*: *inot-d-def itrue-d-def*)
have *3*: $\bigwedge\ \sigma\ i\ j.\ \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty\ \Longrightarrow$
($\sigma\ i\ j \models$ (*suform* (*w iand lfinite*) *n* (*itrue iand lfinite*))) =
($i \neq -\infty \wedge$ ((*upd* σ *n* (λx::*int*. σ (*eint x*) (*eint x*) $\models$ *itrue*)) *i j* $\models$ *w*))
using *subst-suform*[*of n w itrue*] *1 2 assms suform-lfinite-export*[*of w n itrue iand lfinite*]
suform-lfinite-export[*of w n ifalse iand lfinite*] **by** *simp*
have *4*: *state-nl-itl ifalse*
by *simp*
have *5*: ($\bigwedge z.\ z \in$ *fvars ifalse* $\Longrightarrow z \notin$ *bvars w*)
by *simp*
have *6*: $\bigwedge\ \sigma\ i\ j.\ \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty\ \Longrightarrow$
($\sigma\ i\ j \models$ (*suform* (*w iand lfinite*) *n* (*ifalse iand lfinite*))) =
($i \neq -\infty \wedge$ ((*upd* σ *n* (λx::*int*. σ (*eint x*) (*eint x*) $\models$ *ifalse*)) *i j* $\models$ *w*))
using *subst-suform*[*of n w ifalse*] *4 5 assms*
suform-lfinite-export[*of w n itrue iand lfinite*]
suform-lfinite-export[*of w n ifalse iand lfinite*]
by *simp*
have *7*: $\bigwedge\ \sigma\ i\ j\ .\ i \leq j \wedge i \neq \infty \wedge j \neq -\infty\ \Longrightarrow$
($\sigma\ i\ j \models$ (*Ex n. w iand lfinite*)) =
($\exists\, l.\ i \neq -\infty \wedge$ ((*upd* σ *n l*) *i i* $\models$ *w*))
using *assms*(*1*) *lfinite-defs state-nl-itl-defs* **by** *auto*
have *8*: $\bigwedge\ \sigma\ i\ j.\ \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty\ \Longrightarrow$
($i \neq -\infty \wedge$ ((*upd* σ *n* (λx::*int*. σ (*eint x*) (*eint x*) $\models$ *itrue*)) *i j* $\models$ *w*)) =
($i \neq -\infty \wedge$ ((*upd* σ *n* (λx::*int*. *True*)) *i i* $\models$ *w*))
using *state-nl-itl-defs*[*of w*] *assms* **by** *simp*
have *9*: $\bigwedge\ \sigma\ i\ j.\ \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty\ \Longrightarrow$
($i \neq -\infty \wedge$ ((*upd* σ *n* (λx::*int*. σ (*eint x*) (*eint x*) $\models$ *ifalse*)) *i j* $\models$ *w*)) =
($i \neq -\infty \wedge$ ((*upd* σ *n* (λx::*int*. *False*)) *i i* $\models$ *w*))
using *state-nl-itl-defs*[*of w*] *assms* **by** *simp*
have *10*: $\bigwedge\ \sigma\ i\ j.\ \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty\ \Longrightarrow$

$(\exists\, l.\ i \neq -\infty \wedge ((upd\ \sigma\ n\ l)\ i\ i \models w)) \longleftrightarrow$
$(i \neq -\infty \wedge ((upd\ \sigma\ n\ (\lambda x{::}int.\ True))\ i\ i \models w) \vee\ i \neq -\infty \wedge ((upd\ \sigma\ n\ (\lambda x{::}int.\ False))\ i\ i \models w))$
using *assms exists-elim-state-nl-itl-help* **by** *blast*
show *?thesis* **unfolding** *valid-def* **using** *lfinite-defs* **apply** *simp*
by (*meson 3 6 8 9 assms(1) assms(2) exists-elim-state-nl-itl-help state-nl-itl-defs*)
qed

lemma *exists-elim-state-nl-itl-1*:
assumes *state-nl-itl w*
$n \notin bvars\ w$
shows $\vdash$ (*Ex n. w iand lfinite*) *ieqv*
(*suform* (*w iand lfinite*) *n* (*itrue*)) *ior*
(*suform* (*w iand lfinite*) *n* (*ifalse*))
proof −
have *1*: $(\bigwedge z{::}nat.\ z \in fvars\ itrue \Longrightarrow z \notin bvars\ w)$
by (*simp add*: *inot-d-def itrue-d-def*)
have *2*: *state-nl-itl itrue*
by (*simp add*: *inot-d-def itrue-d-def*)
have *3*: $\vdash$ *suform* (*w iand lfinite*) *n* (*itrue iand lfinite*) *ieqv*
suform (*w iand lfinite*) *n* (*itrue*)
using *subst-suform-state*[*of n w itrue*] *assms*
by (*simp add*: *1 2*)
have *4*: $\vdash$ *suform* (*w iand lfinite*) *n* (*ifalse iand lfinite*) *ieqv*
suform (*w iand lfinite*) *n* (*ifalse*)
using *subst-suform-state*[*of n w ifalse*] *assms* **by** (*simp*)
have *5*: $\vdash$ (*suform* (*w iand lfinite*) *n* (*itrue iand lfinite*)) *ior*
(*suform* (*w iand lfinite*) *n* (*ifalse iand lfinite*))
ieqv
(*suform* (*w iand lfinite*) *n* (*itrue*)) *ior*
(*suform* (*w iand lfinite*) *n* (*ifalse*))
using *3 4* **by** *auto*
show *?thesis*
using *5 assms exists-elim-state-nl-itl* **by** *force*
qed

end

7.4 Deep embedding of Fusion Logic with past operators [3].

theory *Fusion-Logic*

imports
NELList-Extras
begin

7.5 Synta and Semanticsx

7.5.1 Primitive formulae

datatype *sf* =
  sfalse-d (*sfalse*)
| *satom-d nat* ((*satom* -))
| *simp-d sf sf* ((- *simp* -) [*26*,*25*] *25*)

datatype *ftr* =
  fform sf
| *ftimp-d ftr ftr* ((- *fimp* -) [*26*,*25*] *25*)
| *next-d sf* ((*tnext* -) [*85*] *85*)

datatype *ptr* =
  pform sf
| *ptimp-d ptr ptr* ((- *pimp* -) [*26*,*25*] *25*)
| *prev-d sf* ((*tprev* -) [*85*] *85*)

datatype *ffe* =
  ftest-d sf ((*ftest* -) [*85*] *85*)
| *fstep-d ftr* ((*fstep* -) [*85*] *85*)
| *fechop-d ffe ffe* ((- *efchop* -) [*26*,*25*] *25*)
| *fchopstar-d ffe* ((*fchopstar* -) [*85*] *85*)
| *ffe-or-d ffe ffe* ((- *feor* -) [*26*,*25*] *25*)

datatype *pfe* =
  ptest-d sf ((*ptest* -) [*85*] *85*)
| *pstep-d ptr* ((*pstep* -) [*85*] *85*)
| *pechop-d pfe pfe* ((- *epchop* -) [*26*,*25*] *25*)
| *pchopstar-d pfe* ((*pchopstar* -) [*85*] *85*)
| *pfe-or-d pfe pfe* ((- *peor* -) [*26*,*25*] *25*)

datatype *fl* =
    false-d (*ifalse*)
  | *atom-d nat* (($-))
  | *iimp-d fl fl* ((- *iimp* -) [*26*,*25*] *25*)
  | *chop-d ffe fl* ((- *chop* -) [*84*,*84*] *83*)
  | *pchop-d fl pfe* ((- *pchop* -) [*84*,*84*] *83*)
  | *until-d fl fl* ((- *until* -) [*84*,*84*] *83*)
  | *since-d fl fl* ((- *since* -) [*84*,*84*] *83*)

7.5.2 Intervals

type-synonym *interval* = *nat set nellist*

fun *semantics-sf* :: [*interval*, *sf*] ⇒ *bool* ((- $\models_s$ -) [*80*,*10*] *10*)
where
  (*M* $\models_s$ *sfalse*) = *False*

$| (M \models_s (satom\text{-}d\ p)) = (p \in (nnth\ M\ 0))$
$| (M \models_s (simp\text{-}d\ F\ G)) = (\ (M \models_s F) \longrightarrow (M \models_s G))$

fun *semantics-ftr* :: [*interval*, *ftr*] ⇒ *bool* ((- $\models_f$ -) [*80*,*10*] *10*)
where
$(M \models_f (fform\ w)) = (\ M \models_s w)$
$| (M \models_f (ftimp\text{-}d\ F\ G)) = (\ (M \models_f F) \longrightarrow (M \models_f G))$
$| (M \models_f (next\text{-}d\ w)) = (\ nlength\ M > 0 \wedge (\ (ndropn\ 1\ M) \models_s w))$

fun *semantics-ptr* :: [*interval*, *ptr*] ⇒ *bool* ((- $\models_p$ -) [*80*,*10*] *10*)
where
$(M \models_p (pform\ w)) = (\ (NNil\ (nnth\ M\ (the\text{-}enat(nlength\ M)))) \models_s w)$
$| (M \models_p (ptimp\text{-}d\ F\ G)) = (\ (M \models_p F) \longrightarrow (M \models_p G))$
$| (M \models_p (prev\text{-}d\ w)) = (\ nfinite\ M \wedge nlength\ M > 0 \wedge (\ (ntaken\ (the\text{-}enat((epred(nlength\ M)))\ )\ M) \models_s w))$

fun *semantics-ffe* :: [*interval*, *ffe*] ⇒ *bool* ((- $\models_x$ -) [*80*,*10*] *10*)
where
$(M \models_x (ftest\text{-}d\ w)) = \ (nlength\ M = 0 \wedge (M \models_s w))$
$| (M \models_x (fstep\text{-}d\ ft)) = (nlength\ M = 1 \wedge (M \models_f ft))$
$| (M \models_x (fechop\text{-}d\ fe1\ fe2)) = (\exists\ k.\ ((ntaken\ k\ M) \models_x fe1) \wedge ((ndropn\ k\ M) \models_x fe2))$
$| (M \models_x (fchopstar\text{-}d\ fe)) = (\exists\ (l{::}\ nat\ nellist).$
$(nnth\ l\ 0) = 0 \wedge nfinite\ l \wedge nidx\ l\ \wedge$
$(enat\ (nlast\ l)) = (nlength\ M) \wedge nfinite\ M \wedge$
$(\forall\ (i{::}nat)\ .\ (\ (enat\ i) < (nlength\ l)) \longrightarrow$
$((nsubn\ M\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ ) \models_x fe)\ )$
$)$
$| (M \models_x (ffe\text{-}or\text{-}d\ fe1\ fe2)) = (\ (M \models_x fe1) \vee (M \models_x fe2))$

fun *semantics-pfe* :: [*interval*, *pfe*] ⇒ *bool* ((- $\models_y$ -) [*80*,*10*] *10*)
where
$(M \models_y (ptest\text{-}d\ w)) = \ (nlength\ M = 0 \wedge (M \models_s w))$
$| (M \models_y (pstep\text{-}d\ pt)) = (nlength\ M = 1 \wedge (M \models_p pt))$
$| (M \models_y (pechop\text{-}d\ pe1\ pe2)) = (\exists\ k.\ ((ntaken\ k\ M) \models_y pe1) \wedge ((ndropn\ k\ M) \models_y pe2))$
$| (M \models_y (pchopstar\text{-}d\ pe)) = (\exists\ (l{::}\ nat\ nellist).$
$(nnth\ l\ 0) = 0 \wedge nfinite\ l \wedge nidx\ l\ \wedge$
$(enat\ (nlast\ l)) = (nlength\ M) \wedge nfinite\ M \wedge$
$(\forall\ (i{::}nat)\ .\ (\ (enat\ i) < (nlength\ l)) \longrightarrow$
$((nsubn\ M\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ ) \models_y pe)\ )$
$)$
$| (M \models_y (pfe\text{-}or\text{-}d\ pe1\ pe2)) = (\ (M \models_y pe1) \vee (M \models_y pe2))$

fun *semantics-fl* :: [*interval*, *nat*, *fl*] ⇒ *bool* ((- - $\models$ -) [*80*,*10*] *10*)
where
$(M\ i \models ifalse) = False$
$| (M\ i \models \$p) = (\ p \in (nnth\ M\ i)\ )$
$| (M\ i \models fl1\ iimp\ fl2) = (\ (M\ i \models fl1) \longrightarrow (M\ i \models fl2)\ )$
$| (M\ i \models fe\ chop\ fl) = (\ \exists\ k.\ i \leq k \wedge k \leq nlength\ M \wedge ((nsubn\ M\ i\ k)\ \models_x fe) \wedge (M\ k \models fl))$

$|\ (M\ i \models \mathit{fl}\ \mathit{pchop}\ \mathit{pe}) = (\exists\ k.\ k \leq i \wedge (M\ k \models \mathit{fl}) \wedge ((\mathit{nsubn}\ M\ k\ i)\ \models_y \mathit{pe})\ )$
$|\ (M\ i \models \mathit{fl1}\ \mathit{until}\ \mathit{fl2}) =$
$\quad (\exists\ k.\ i \leq k \wedge k \leq \mathit{nlength}\ M \wedge (M\ k \models \mathit{fl2}) \wedge (\forall\ n.\ i < n \wedge n < k \longrightarrow (M\ n \models \mathit{fl1})))$
$|\ (M\ i \models \mathit{fl1}\ \mathit{since}\ \mathit{fl2}) =$
$\quad (\exists\ k.\ k \leq i \wedge (M\ k \models \mathit{fl2}) \wedge (\forall\ n.\ k < n \wedge n < i \longrightarrow (M\ n \models \mathit{fl1})))$

7.5.3 Derived Boolean Operators

definition *ffalse*
where $\mathit{ffalse} \equiv \mathit{fform}\ \mathit{sfalse}$

definition *pfalse*
where $\mathit{pfalse} \equiv \mathit{pform}\ \mathit{sfalse}$

definition *inot-d* ((*inot* -) [*90*] *90*)
where
$\mathit{inot}\ f \equiv f\ \mathit{iimp}\ \mathit{ifalse}$

definition *fnot-d* ((*fnot* -) [*90*] *90*)
where
$\mathit{fnot}\ f \equiv f\ \mathit{fimp}\ \mathit{ffalse}$

definition *pnot-d* ((*pnot* -) [*90*] *90*)
where
$\mathit{pnot}\ f \equiv f\ \mathit{pimp}\ \mathit{pfalse}$

definition *snot-d* ((*snot* -) [*90*] *90*)
where
$\mathit{snot}\ f \equiv f\ \mathit{simp}\ \mathit{sfalse}$

definition *itrue-d* (*itrue*)
where
$\mathit{itrue} \equiv \mathit{inot}\ \ \mathit{ifalse}$

definition *ftrue-d* (*ftrue*)
where
$\mathit{ftrue} \equiv \mathit{fnot}\ \ \mathit{ffalse}$

definition *ptrue-d* (*ptrue*)
where
$\mathit{ptrue} \equiv \mathit{pnot}\ \ \mathit{pfalse}$

definition *strue-d* (*strue*)
where
$\mathit{strue} \equiv \mathit{snot}\ \ \mathit{sfalse}$

definition *ior-d* ((- *ior* -) [*31,30*] *30*)
where
$f\ \mathit{ior}\ g \equiv (\mathit{inot}\ f)\ \mathit{iimp}\ g$

definition *ftor-d* ((- *ftor* -) [*31*,*30*] *30*)
where
f ftor g ≡ (*fnot f*) *fimp g*

definition *ptor-d* ((- *ptor* -) [*31*,*30*] *30*)
where
f ptor g ≡ (*pnot f*) *pimp g*

definition *sor-d* ((- *sor* -) [*31*,*30*] *30*)
where
f sor g ≡ (*snot f*) *simp g*

definition *iand-d* ((- *iand* -) [*36*,*35*] *35*)
where
f iand g ≡ *inot* (*inot f ior inot g*)

definition *ftand-d* ((- *ftand* -) [*36*,*35*] *35*)
where
f ftand g ≡ *fnot* (*fnot f ftor fnot g*)

definition *ptand-d* ((- *ptand* -) [*36*,*35*] *35*)
where
f ptand g ≡ *pnot* (*pnot f ptor pnot g*)

definition *sand-d* ((- *sand* -) [*36*,*35*] *35*)
where
f sand g ≡ *snot* (*snot f sor snot g*)

definition *ieqv-d* ((- *ieqv* -) [*21*,*20*] *20*)
where
f ieqv g ≡ ((*f iimp g*) *iand* (*g iimp f*))

definition *fteqv-d* ((- *fteqv* -) [*21*,*20*] *20*)
where
f fteqv g ≡ ((*f fimp g*) *ftand* (*g fimp f*))

definition *pteqv-d* ((- *pteqv* -) [*21*,*20*] *20*)
where
f pteqv g ≡ ((*f pimp g*) *ptand* (*g pimp f*))

definition *seqv-d* ((- *seqv* -) [*21*,*20*] *20*)
where
f seqv g ≡ ((*f simp g*) *sand* (*g simp f*))

7.5.4 Future fusion expressions

definition *ffe-true-d* (*ffetrue*)
where *ffetrue* ≡ *fchopstar* (*fstep* (*fform strue*))

7.5.5 Past fusion expressions

definition *pfe-true-d* (*pfetrue*)
where *pfetrue* ≡ *pchopstar* (*pstep* (*pform strue*))

7.5.6 Next and Previous

definition *next-d* ((*next* -) [*88*] *87*)
where
next f ≡ (*fstep ftrue*) *chop f*

definition *prev-d* ((*prev* -) [*88*] *87*)
where
prev f ≡ *f pchop* (*pstep ptrue*)

7.5.7 Box and Diamond Operators

definition *diamond-d* ((*diamond* -) [*88*] *87*)
where
diamond f ≡ *itrue until f*

definition *box-d* ((*box* -) [*88*] *87*)
where
box f ≡ *inot* (*diamond* (*inot f*))

definition *sdiamond-d* ((*sdiamond* -) [*88*] *87*)
where
sdiamond f ≡ *next* (*diamond f*)

definition *sbox-d* ((*sbox* -) [*88*] *87*)
where
sbox f ≡ *inot* (*sdiamond* (*inot f*))

definition *pdiamond-d* ((*pdiamond* -) [*88*] *87*)
where
pdiamond f ≡ *itrue since f*

definition *pbox-d* ((*pbox* -) [*88*] *87*)
where
pbox f ≡ *inot* (*pdiamond* (*inot f*))

definition *psdiamond-d* ((*psdiamond* -) [*88*] *87*)
where
psdiamond f ≡ *prev* (*pdiamond f*)

definition *psbox-d* ((*psbox* -) [*88*] *87*)
where
psbox f ≡ *inot* (*psdiamond* (*inot f*))

7.5.8 more, empty, skip and wnext

definition *more-d* (*more*)
where
more ≡ *next itrue*

definition *empty-d* (*empty*)
where
empty ≡ *inot more*

definition *skip-d* (*skip*)
where
skip ≡ *next empty*

definition *wnext-d* ((*wnext* -) [*88*] *87*)
where
wnext f ≡ *inot* (*next* (*inot f*))

7.5.9 pmore, pempty, pskip and wprev

definition *pmore-d* (*pmore*)
where
pmore ≡ *prev itrue*

definition *pempty-d* (*pempty*)
where
pempty ≡ *inot pmore*

definition *pskip-d* (*pskip*)
where
pskip ≡ *prev pempty*

definition *wprev-d* ((*wprev* -) [*88*] *87*)
where
wprev f ≡ *inot* (*prev* (*inot f*))

7.5.10 ssince and suntil

definition *suntil-d* ((- *suntil* -) [*84,84*] *83*)
where
f suntil g ≡ *next* (*f until g*)

definition *ssince-d* ((- *ssince* -) [*84,84*] *83*)
where
f ssince g ≡ *prev* (*f since g*)

7.5.11 ifinite and inf

definition *finite-d* (*ifinite*)
where
ifinite ≡ *ffetrue chop empty*

definition *inf-d* (*inf*)
where
inf ≡ *inot ifinite*

7.5.12 future and past len

primrec *flen-d* ((*flen* -) [*88*] *87*)
where
flen-0 : (*flen-d* *0*) = *empty*
| *flen-Suc*: (*flen-d* (*Suc n*)) = ((*fstep* (*fform strue*)) *chop* (*flen-d* *n*))

primrec *plen-d* ((*plen* -) [*88*] *87*)
where
plen-0 : (*plen-d 0*) = *pempty*
| *plen-Suc*: (*plen-d* (*Suc n*)) = ((*plen-d n*) *pchop* (*pstep* (*pform strue*)))

7.5.13 Validity

definition *valid* :: *fl* ⇒ *bool* ((⊢ -) *5*)
where (⊢ *f*) = (∀ *M i*. *i* ≤ *nlength M* ⟶ (*M i* ⊨ *f*))

definition *valid-fin* :: *fl* ⇒ *bool* ((⊢$_f$ -) *5*)
where (⊢$_f$ *f*) = (∀ *M i*. *i* ≤ *nlength M* ∧ *nfinite M* ⟶ (*M i* ⊨ *f*))

7.5.14 Defs

lemma *inot-defs* [*simp*]:
shows (*M i* ⊨ *inot f*) = *Not* (*M i* ⊨ *f*)
by (*simp add*: *inot-d-def*)

lemma *ior-defs* [*simp*]:
(*M i* ⊨ *f1 ior f2*) = ((*M i* ⊨ *f1*) ∨ (*M i* ⊨ *f2*))
using *semantics-fl.simps*
by (*metis inot-defs ior-d-def*)

lemma *iand-defs* [*simp*]:
(*M i*⊨ *f1 iand f2*) = ((*M i* ⊨ *f1*) ∧ (*M i*⊨ *f2*))
using *iand-d-def inot-defs ior-defs* **by** *presburger*

lemma *ieqv-defs* [*simp*]:
(*M i* ⊨ *f1 ieqv f2*) = ((*M i* ⊨ *f1*) = (*M i* ⊨ *f2*))
using *iand-defs ieqv-d-def* **by** *auto*

lemma *itrue-defs* [*simp*]:
(*M i* ⊨ *itrue*)
by (*simp add*: *inot-d-def itrue-d-def*)

lemma *ftrue-defs* [*simp*]:
(*M* ⊨$_f$ *ftrue*)

by (*simp add*: *fnot-d-def ftrue-d-def*)

lemma *ptrue-defs* [*simp*]:
(*M* $\models_p$ *ptrue*)
by (*simp add*: *pnot-d-def ptrue-d-def*)

lemma *fstep-ftrue-defs* [*simp*]:
(*M* $\models_x$ *fstep ftrue*) = (*nlength M* = *1*)
by *simp*

lemma *pstep-ptrue-defs* [*simp*]:
(*M* $\models_y$ *pstep ptrue*) = (*nlength M* = *1*)
by *simp*

lemma *next-defs*:
assumes *i* ≤ *nlength M*
shows (*M i* ⊨ *next f*) = (*i*< *nlength M* ∧ (*M* (*Suc i*) ⊨ *f*))
proof −
have *1*: (*M i* ⊨ *next f*) = (*M i* ⊨ *fstep ftrue chop f*)
unfolding *next-d-def* **by** *blast*
have *2*: (*M i* ⊨ *fstep ftrue chop f*) =
(∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ *nlength* (*nsubn M i k*) = *1* ∧ (*M k* ⊨ *f*))
by *simp*
have *3*: (∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ *nlength* (*nsubn M i k*) = *1* ∧ (*M k* ⊨ *f*)) =
(∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ *min* (*enat k* − *enat i*) (*nlength M* − *enat i*) = *1* ∧ (*M k* ⊨ *f*))
using *assms nsubn-nlength*[*of M i* -] **by** *presburger*
have *4*: (∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ *min* (*enat k* − *enat i*) (*nlength M* − *enat i*) = *1* ∧ (*M k* ⊨ *f*)) =
(*i*< *nlength M* ∧ (*M* (*Suc i*) ⊨ *f*))
using *assms* **apply** *auto*
apply (*metis diff-is-0-eq*′ *enat-ord-simps*(*1*) *idiff-enat-enat min.idem order-le-less zero-enat-def zero-one-enat-neq*(*1*)
apply (*metis One-nat-def Suc-pred diff-diff-cancel diff-le-self enat-minus-mono1 idiff-enat-enat less-eq-Suc-le*
min.absorb1
one-enat-def the-enat.simps)
by (*metis Suc-ile-eq add-diff-cancel-left*′ *add-diff-cancel-right*′ *diff-le-self enat-minus-mono1 idiff-enat-enat*
min.absorb1
one-enat-def plus-1-eq-Suc)
show *?thesis* **using** *1 2 3 4* **by** *blast*
qed

lemma *prev-defs*:
assumes *i* ≤ *nlength M*
shows (*M i* ⊨ *prev f*) = (*i*>*0* ∧ (*M* (*i*−*1*) ⊨ *f*))
proof −
have *1*: (*M i* ⊨ *prev f*) = (*M i* ⊨ *f pchop pstep ptrue*)
using *assms* **unfolding** *prev-d-def* **by** *blast*
have *2*: (*M i* ⊨ *f pchop pstep ptrue*) = (∃ *k*≤*i*. (*M k* ⊨ *f*) ∧ *nlength* (*nsubn M k i*) = *1*)
by *simp*

have *3*: $(\exists k{\leq}i.\ (M\ k\ \models f) \wedge nlength\ (nsubn\ M\ k\ i) = 1) =$
$(\exists k{\leq}i.\ (M\ k\ \models f) \wedge min\ (enat\ i - enat\ k)\ (nlength\ M - enat\ k) = 1)$
using *assms nsubn-nlength*[*of M - i*] **by** *presburger*

have *4*: $(\exists k{\leq}i.\ (M\ k\ \models f) \wedge min\ (enat\ i - enat\ k)\ (nlength\ M - enat\ k) = 1) =$
$(i{>}0 \wedge (M\ (i{-}1) \models f))$
using *assms*
apply *auto*
apply (*metis bot-nat-0.extremum-unique bot-nat-0.not-eq-extremum diff-self-eq-0 min-enat-simps*(*3*) *zero-enat-def*
zero-one-enat-neq(*1*))
apply (*metis One-nat-def diff-diff-cancel enat.inject enat-minus-mono1 idiff-enat-enat min.absorb1*
one-enat-def)
by (*metis One-nat-def diff-diff-cancel diff-le-self enat-minus-mono1 idiff-enat-enat less-eq-Suc-le min-def*
one-enat-def)
show *?thesis* **using** *1 2 3 4* **by** *blast*
qed

lemma *diamond-defs*:
assumes $i \leq nlength\ M$
shows $(M\ i \models diamond\ f) = (\exists\ k.\ i \leq k \wedge k \leq nlength\ M \wedge (M\ k \models f))$
using *assms*
unfolding *diamond-d-def*
by *simp*

lemma *box-defs*:
assumes $i \leq nlength\ M$
shows $(M\ i \models box\ f) = (\forall\ k.\ i \leq k \wedge k \leq nlength\ M \longrightarrow (M\ k \models f))$
using *assms*
unfolding *box-d-def*
apply (*simp add*: *diamond-defs*)
by *blast*

lemma *sdiamond-defs*:
assumes $i \leq nlength\ M$
shows $(M\ i \models sdiamond\ f) = (\exists\ k.\ i < k \wedge k \leq nlength\ M \wedge (M\ k \models f))$
proof −
have *1*: $(M\ i \models sdiamond\ f) = (\ M\ i\ \models next\ (diamond\ f))$
using *assms* **unfolding** *sdiamond-d-def* **by** *blast*
have *2*: $(\ M\ i\ \models next\ (diamond\ f)) =$
$(enat\ i < nlength\ M \wedge (\exists k{\geq}Suc\ i.\ enat\ k \leq nlength\ M \wedge (M\ k\ \models f)))$
using *assms diamond-defs*[*of Suc i M f*] *next-defs*[*of i M diamond f*]
using *Suc-ile-eq* **by** *blast*
have *3*: $(enat\ i < nlength\ M \wedge (\exists k{\geq}Suc\ i.\ enat\ k \leq nlength\ M \wedge (M\ k\ \models f))) =$
$(\exists\ k.\ i < k \wedge k \leq nlength\ M \wedge (M\ k \models f))$
using *assms*
by (*meson Suc-le-eq dual-order.strict-trans1 enat-ord-simps*(*2*))
show *?thesis*
using *1 2 3* **by** *presburger*

qed

lemma *sbox-defs*:
assumes $i \leq$ *nlength M*
shows $(M\ i \models sbox\ f) = (\forall\ k.\ i < k \wedge k \leq nlength\ M \longrightarrow (M\ k \models f))$
using *assms*
unfolding *sbox-d-def*
apply (*simp add*: *sdiamond-defs*)
by *blast*

lemma *pdiamond-defs*:
assumes $i \leq$ *nlength M*
shows $(M\ i \models pdiamond\ f) = (\exists\ k.\ k \leq i \wedge (M\ k \models f))$
using *assms* **unfolding** *pdiamond-d-def*
by *simp*

lemma *pbox-defs*:
assumes $i \leq$ *nlength M*
shows $(M\ i \models pbox\ f) = (\forall\ k.\ k \leq i \longrightarrow (M\ k \models f))$
using *assms*
unfolding *pbox-d-def*
by (*simp add*: *pdiamond-defs*)

lemma *psdiamond-defs*:
assumes $i \leq$ *nlength M*
shows $(M\ i \models psdiamond\ f) = (\exists\ k.\ k < i \wedge (M\ k \models f))$
proof −
have *1*: $(M\ i \models psdiamond\ f) = (M\ i \models prev\ (pdiamond\ f))$
unfolding *psdiamond-d-def* **by** *blast*
have *2*: $(M\ i \models prev\ (pdiamond\ f)) =$
$(0 < i \wedge (\exists k \leq i - 1.\ M\ k \models f))$
using *assms pdiamond-defs*[*of i−1 M f*] *prev-defs*[*of i M pdiamond f*]
using *pdiamond-d-def* **by** *force*
have *3*: $(0 < i \wedge (\exists k \leq i - 1.\ M\ k \models f)) = (\exists\ k.\ k < i \wedge (M\ k \models f))$
apply *simp*
by (*metis Suc-pred less-Suc-eq-le neq0-conv not-less-zero*)
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *psbox-defs*:
assumes $i \leq$ *nlength M*
shows $(M\ i \models psbox\ f) = (\forall\ k.\ k < i \longrightarrow (M\ k \models f))$
using *assms*
unfolding *psbox-d-def*
by (*simp add*: *psdiamond-defs*)

lemma *skip-defs*:
assumes $i \leq$ *nlength* σ
shows $(\sigma\ i \models skip\ ) \longleftrightarrow nlength\ \sigma - i = 1$
proof −
have *1*: $(\sigma\ i \models skip\ ) = (\sigma\ i \models next\ inot\ (next\ itrue))$
unfolding *skip-d-def empty-d-def more-d-def* **by** *blast*
have *2*: $(\sigma\ i \models next\ inot\ (next\ itrue)) =$
$(enat\ i < nlength\ \sigma \wedge (\sigma\ (Suc\ i) \models inot\ (next\ itrue)))$
using *next-defs*[*of i* σ *inot* (*next itrue*)] *assms* **by** *blast*
have *3*: $(enat\ i < nlength\ \sigma \wedge (\sigma\ (Suc\ i) \models inot\ (next\ itrue))) =$
$(enat\ i < nlength\ \sigma \wedge \neg((enat\ (Suc\ i) < nlength\ \sigma \wedge (\sigma\ (Suc\ (Suc\ i)) \models itrue))))$
using *next-defs*[*of Suc i* σ *itrue*] *assms* **using** *Suc-ile-eq* **by** *force*
have *4*: $(enat\ i < nlength\ \sigma \wedge \neg((enat\ (Suc\ i) < nlength\ \sigma \wedge (\sigma\ (Suc\ (Suc\ i)) \models itrue)))) =$
$(nlength\ \sigma - i = 1)$
using *assms* **apply** *auto*
apply (*metis add-diff-cancel-right′ eSuc-enat idiff-enat-enat ileI1 nless-le one-enat-def plus-1-eq-Suc*)
using *dual-order.order-iff-strict zero-enat-def* **apply** *fastforce*
by (*metis add.commute dual-order.strict-iff-order enat.distinct*(*2*) *enat-add-sub-same le-iff-add one-enat-def plus-1-eq-Suc*
plus-enat-simps(*1*))
show *?thesis* **using** *1 2 3 4* **by** *blast*
qed

lemma *pmore-defs*:
assumes $i \leq$ *nlength* σ
shows $(\sigma\ i \models pmore) \longleftrightarrow i > 0$
using *assms* **unfolding** *pmore-d-def*
using *prev-defs*[*of*]
by *simp*

lemma *pempty-defs*:
assumes $i \leq$ *nlength* σ
shows $(\sigma\ i \models pempty) \longleftrightarrow i = 0$
using *assms*
pmore-defs
by (*simp add*: *pempty-d-def*)

lemma *pskip-defs*:
assumes $i \leq$ *nlength* σ
shows $(\sigma\ i \models pskip\ ) \longleftrightarrow i = 1$
proof −
have *1*: $(\sigma\ i \models pskip\ ) = (\sigma\ i \models prev\ inot\ (prev\ itrue))$
unfolding *pskip-d-def pempty-d-def pmore-d-def* **by** *blast*
have *2*: $(\sigma\ i \models prev\ inot\ (prev\ itrue)) = (0 < i \wedge (\sigma\ (i - 1) \models inot\ (prev\ itrue)))$
using *prev-defs*[*of i* σ *inot* (*prev itrue*)] *assms* **by** *blast*
have *3*: $(0 < i \wedge (\sigma\ (i - 1) \models inot\ (prev\ itrue))) =$
$(0 < i \wedge \neg((0 < i - 1 \wedge (\sigma\ (i - 1 - 1) \models itrue))))$

using *prev-defs*[*of* $i-1$ σ *itrue*] *assms*
by (*metis One-nat-def Suc-ile-eq Suc-pred dual-order.order-iff-strict inot-defs*)
have *4*: $(0 < i \wedge \neg((0 < i - 1 \wedge (\sigma\ (i - 1 - 1) \models itrue)))) =$
$(i = 1)$
using *assms* **by** *auto*
show *?thesis* **using** *1 2 3 4* **by** *blast*
qed

lemma *pbox-itrue*:
assumes $i \leq nlength\ \sigma$
shows $(\sigma\ i \models pbox\ itrue)$
using *assms pbox-defs*
by *simp*

7.6 Time reversal

primrec *ft-to-pt* :: *ftr* ⇒ *ptr*
where
ft-to-pt (*fform w*) = (*tprev w*)
| *ft-to-pt* (*ft1 fimp ft2*) = ((*ft-to-pt ft1*) *pimp* (*ft-to-pt ft2*))
| *ft-to-pt* (*tnext w*) = (*pform w*)

primrec *rvft* :: *ftr* ⇒ *ptr*
where
rvft (*fform w*) = (*pform w*)
| *rvft* (*ft1 fimp ft2*) = ((*rvft ft1*) *pimp* (*rvft ft2*))
| *rvft* (*tnext w*) = (*tprev w*)

primrec *pt-to-ft* :: *ptr* ⇒ *ftr*
where
pt-to-ft (*pform w*) = (*tnext w*)
| *pt-to-ft* (*pt1 pimp pt2*) = ((*pt-to-ft pt1*) *fimp* (*pt-to-ft pt2*))
| *pt-to-ft* (*tprev w*) = (*fform w*)

primrec *rvpt* :: *ptr* ⇒ *ftr*
where
rvpt (*pform w*) = (*fform w*)
| *rvpt* (*pt1 pimp pt2*) = ((*rvpt pt1*) *fimp* (*rvpt pt2*))
| *rvpt* (*tprev w*) = (*tnext w*)

primrec *rvffe*:: *ffe* ⇒ *pfe*
where
rvffe (*ftest w*) = (*ptest w*)
| *rvffe* (*fstep ft*) = (*pstep* (*rvft ft*))
| *rvffe* (*ffe1 efchop ffe2*) = ((*rvffe ffe2*) *epchop* (*rvffe ffe1*))
| *rvffe* (*fchopstar ffe*) = (*pchopstar* (*rvffe ffe*))
| *rvffe* (*ffe1 feor ffe2*) = ((*rvffe ffe1*) *peor* (*rvffe ffe2*))

primrec *rvpfe*:: *pfe* ⇒ *ffe*

```
 where
   rvpfe (ptest w)          = (ftest w)
|  rvpfe (pstep pt)          = (fstep (rvpt pt))
|  rvpfe (pfe1 epchop pfe2) = ((rvpfe pfe2) efchop (rvpfe pfe1))
|  rvpfe (pchopstar pfe)    = (fchopstar (rvpfe pfe))
|  rvpfe (pfe1 peor pfe2)   = ((rvpfe pfe1) feor (rvpfe pfe2))


primrec rv :: fl ⇒ fl
where
   rv ifalse        = ifalse
|  rv ($p)          = $p
|  rv (f iimp g)    = ((rv f) iimp  (rv g))
|  rv (ffe chop g)  = ((rv g) pchop (rvffe ffe))
|  rv (f pchop pfe) = ((rvpfe pfe)  chop (rv f))
|  rv (f until g)   = ((rv f) since (rv g))
|  rv (f since g)   = ((rv f) until (rv g))


lemma sf-sem:
  (M ⊨s w) = ( (NNil (nnth M 0)) ⊨s w)
proof (induction w arbitrary: M)
case sfalse-d
then show ?case by simp
next
case (satom-d x)
then show ?case by (simp add: nnth-NNil)
next
case (simp-d w1 w2)
then show ?case by simp
qed

lemma fform-sem:
  (M ⊨f (fform w)) = ( (NNil (nnth M 0)) ⊨f (fform w))
using sf-sem by auto

lemma pform-sem:
assumes nfinite M
shows  (M ⊨p (pform w)) = ( (NNil (nnth M (the-enat (nlength M)))) ⊨p (pform w))
by (simp add: nnth-NNil)

lemma ft-to-pt-sem:
assumes nfinite M
        nlength M = 1
shows  (M ⊨p (ft-to-pt ft)) = (M ⊨f ft)
using assms
proof (induction ft arbitrary: M)
case (fform x)
then show ?case apply simp
by (metis nfirst-eq-nnth-zero sf-sem)
```

next
case (*ftimp-d ft1 ft2*)
then show *?case* **by** *simp*
next
case (*next-d x*)
then show *?case* **apply** *simp*
by (*metis eSuc-enat enat-le-plus-same*(*2*) *ndropn-all nnth-nlast*
plus-1-eSuc(*2*))
qed

lemma *rvft-sem*:
assumes *nfinite M*
nlength M = *1*
shows (*M* ⊨$_p$ (*rvft ft*)) = ((*nrev M*) ⊨$_f$ *ft*)
using *assms*
proof (*induction ft arbitrary*: *M*)
case (*fform x*)
then show *?case* **apply** *simp*
by (*metis enat-0-iff*(*1*) *minus-nat.diff-0 nnth-nrev sf-sem*
zero-le)
next
case (*ftimp-d ft1 ft2*)
then show *?case* **by** *simp*
next
case (*next-d x*)
then show *?case* **apply** *simp*
by (*metis eSuc-enat enat-le-plus-same*(*2*) *ndropn-all nlast-nrev*
nlength-nrev plus-1-eSuc(*2*))
qed

lemma *pt-to-ft-sem*:
assumes *nfinite M*
nlength M = *1*
shows (*M* ⊨$_f$ (*pt-to-ft pt*)) = (*M* ⊨$_p$ *pt*)
using *assms*
proof (*induction pt arbitrary*: *M*)
case (*pform x*)
then show *?case* **apply** *simp*
by (*simp add*: *ndropn-all nnth-nlast one-enat-def*)
next
case (*ptimp-d pt1 pt2*)
then show *?case* **by** *simp*
next
case (*prev-d x*)
then show *?case* **apply** *simp*
by (*metis nfirst-eq-nnth-zero sf-sem*)
qed

lemma *rvpt-sem*:

assumes *nfinite M*
nlength M = 1
shows $(M \models_f (rvpt\ pt)) = ((nrev\ M) \models_p pt)$
using *assms*
proof (*induction pt arbitrary*: *M*)
case (*pform x*)
then show *?case* **apply** *simp*
by (*metis diff-is-0-eq le-numeral-extra*(*4*) *nnth-nrev*
one-enat-def sf-sem the-enat.simps)
next
case (*ptimp-d pt1 pt2*)
then show *?case* **by** *simp*
next
case (*prev-d x*)
then show *?case* **apply** *simp*
by (*metis One-nat-def ndropn-nlast nfinite-nrev nfirst-nrev*
one-enat-def the-enat.simps)
qed

lemma *rvffe-sem*:
assumes *nfinite M*
shows $(M \models_y (rvffe\ fe)) = ((nrev\ M) \models_x fe)$
using *assms*
proof (*induction fe arbitrary*: *M*)
case (*ftest-d x*)
then show *?case* **apply** *simp*
by (*metis nrev-NNil ntake-0 ntake-all zero-le*)
next
case (*fstep-d x*)
then show *?case*
apply *simp*
using *rvft-sem* **by** *auto*
next
case (*fechop-d fe1 fe2*)
then show *?case* **apply** *auto*
apply (*metis le-less ndropn-all nfinite-conv-nlength-enat*
not-le-imp-less nrev-ndropn nrev-ntaken ntaken-all)
by (*metis* (*no-types*, *lifting*) *diff-zero le-less ndropn-0*
ndropn-all ndropn-nlast nfinite-nrev nlength-nrev
not-le-imp-less nrev-ndropn nrev-nrev nrev-ntaken
ntaken-all)
next
case (*fchopstar-d fe*)
then show *?case*
proof −
have *c1*: $(M \models_y rvffe\ (fchopstar\ fe)) =$
$(\exists l.\ nnth\ l\ 0 = 0 \wedge$
$nfinite\ l \wedge$
$nidx\ l \wedge$
$enat\ (nlast\ l) = nlength\ M \wedge$

```
        nfinite M ∧ (∀ i. enat i < nlength l ⟶ ((nsubn M (nnth l i) (nnth l (Suc i)))  ⊨y rvffe fe)))
    by simp
  have c2: (nrev M  ⊨x fchopstar fe ) =
          (∃ l. nnth l 0 = 0 ∧
        nfinite l ∧
        nidx l ∧
        enat (nlast l) = nlength M ∧
        nfinite (nrev M) ∧ (∀ i. enat i < nlength l ⟶ ((nsubn (nrev M) (nnth l i) (nnth l (Suc i)))  ⊨x
fe)))
    by simp
  have c3: (M  ⊨y rvffe (fchopstar fe )) ⟹ (nrev M  ⊨x fchopstar fe )
   proof −
    assume a0: (M  ⊨y rvffe (fchopstar fe ))
    show (nrev M  ⊨x fchopstar fe )
    proof −
     obtain l where c31: nnth l 0 = 0 ∧
        nfinite l ∧
        nidx l ∧
        enat (nlast l) = nlength M ∧
        nfinite M ∧ (∀ i. enat i < nlength l ⟶ ((nsubn M (nnth l i) (nnth l (Suc i)))  ⊨y rvffe fe))
     using a0 c1 by blast
    have c32: nfinite (nrev M)
     by (simp add: fchopstar-d.prems nfinite-nrev)
    have c33: (∀ i. enat i < nlength l ⟶ ( (nrev (nsubn M (nnth l i) (nnth l (Suc i)) ) )  ⊨x  fe))
    by (meson c31 fchopstar-d.IH nsubn-nfinite)
    have c34: ⋀ j. j < nlength l ⟹ (nnth l j) ≤ (nnth l (Suc j))
    using Suc-ile-eq c31 nidx-less-eq by auto
    have c35: ⋀j. j ≤ nlength l ⟹ (nnth l j) ≤ nlength M
    by (metis c31 enat-ord-simps(1) nidx-all-le-nlast)
    have c36: ⋀j. j < nlength l ⟹
                (nrev (nsubn M (nnth l j) (nnth l (Suc j)) ) ) =
                nsubn (nrev M) ((the-enat(nlength M)) −(nnth l (Suc j)))  ((the-enat(nlength M)) −(nnth l
j))
     by (simp add: Suc-ile-eq c31 c34 c35 nrev-nsubn)
    have c360: ∃ x. (nlength M) = (enat x)
     by (metis c31)
    obtain x where  c361: (nlength M) = (enat x)
      using c360 by blast
    have c362: ∃ y. (nlength l) = (enat y)
    using c31 nfinite-conv-nlength-enat by auto
    obtain y where  c363: (nlength l) = (enat y)
      using c362 by blast
    have c364: ⋀j. j ≤ (nlength l) ⟹ (nnth (nrev l) j) = nnth l (y − j)
       using nnth-nrev[of l] by (simp add: c31 c363)
    have c365: ⋀j. j ≤ (nlength l) ⟹ nnth l (y − j) ≤ x
      using c35 c361 c363 by auto

    have c37: ⋀j. j < nlength (nrev l) ⟹
                (x − (nnth (nrev l) j) <
                  x − (nnth (nrev l) (Suc j)))
```

by (*metis* (*no-types*, *lifting*) *ext Suc-diff-Suc c31 c363 c365 diff-le-self diff-less-mono2 dual-order.strict-iff-order eSuc-enat enat-ord-simps*(*1*)
ileI1 less-le-not-le nidx-expand nlength-nrev nnth-nrev the-enat.simps)

have *c38*: ((*the-enat*(*nlength M*)) − *nnth* (*nrev l*) *0*) = *0*
by (*metis c31 dual-order.strict-iff-order nfirst-eq-nnth-zero*
nfirst-nrev not-gr-zero the-enat.simps zero-less-diff)
have *c39*: ((*the-enat*(*nlength M*)) − (*nlast* (*nrev l*))) = *nlength M*
by (*metis c31 minus-nat.diff-0 nfirst-eq-nnth-zero nlast-nrev*
the-enat.simps)
have *c40*: *nridx* (λ*x y*. ((*the-enat*(*nlength M*)) − *x*) < ((*the-enat*(*nlength M*)) − *y*)) (*nrev l*)
by (*simp add*: *Suc-ile-eq c361 c37 nridx-expand*)
have *c41*: ∃ *l1*. *nlength l1* = *nlength l* ∧ (∀ *i*. *i*≤ *nlength l1* ⟶ (*nnth l1 i*) = ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)))
by (*metis nlength-nmap nlength-nrev nnth-nmap*)
obtain *l1* **where** *c42*: *nlength l1* = *nlength l* ∧ (∀ *i*. *i*≤ *nlength l1* ⟶ (*nnth l1 i*) = ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)))
using *c41* **by** *blast*
have *c43*: *nnth l1 0* = *0*
using *c38 c42 enat-defs*(*1*) **by** *fastforce*
have *c44*: *nfinite l1*
by (*metis c31 c42 nfinite-conv-nlength-enat*)
have *c45*: *enat* (*nlast l1*) = *nlength M*
by (*metis c39 c42 c44 dual-order.refl nfinite-conv-nlength-enat*
nlength-nrev nnth-nlast the-enat.simps)
have *c46*: *nidx l1*
using *Suc-ile-eq c361 c37 c42 nidx-expand* **by** *auto*
have *c47*: ⋀*j*. *j* ≤ *nlength l* ⟹
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *j*)) =
((*the-enat*(*nlength M*)) − (*nnth l* ((*the-enat* (*nlength l*))− *j*)))
by (*simp add*: *c31 nnth-nrev*)

have *c48*: ⋀*k*. *k* ≤ *nlength l* ⟹
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− *k*))) =
((*the-enat*(*nlength M*)) − (*nnth l k*))
by (*metis c31 nfinite-nrev nlength-nrev nnth-nrev*
nrev-nrev)
have *c49*: ⋀*k*. *k* < *nlength l1* ⟹
nsubn (*nrev M*) ((*the-enat*(*nlength M*)) −(*nnth l* (*Suc k*))) ((*the-enat*(*nlength M*)) −(*nnth l k*)) =
nsubn (*nrev M*) ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− (*Suc k*))))
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− *k*)))
by (*simp add*: *Suc-ile-eq c42 c48*)
have *c500*: (∀ *i*. *enat i* < *nlength l* ⟶
(*nsubn* (*nrev M*) ((*the-enat*(*nlength M*)) −(*nnth l* (*Suc i*)))
((*the-enat*(*nlength M*)) −(*nnth l i*)) ⊨$_x$ *fe*))
using *c33 c36* **by** *auto*
have *c50*: (∀ *i*. *enat i* < *nlength l1* ⟶
((*nsubn* (*nrev M*) ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*))
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) (*Suc i*)))) ⊨$_x$ *fe*))

by (*metis Suc-diff-Suc Suc-ile-eq c363 c364 c42 c500 diff-le-self enat-ord-simps(1) less-le-not-le*)

have *c51*: (∀ *i*. *enat i* < *nlength l1* ⟶
((*nsubn* (*nrev M*) (*nnth l1 i*) (*nnth l1* (*Suc i*))) ⊨$_x$ *fe*))
by (*simp add*: *Suc-ile-eq c42 c50*)
show *?thesis*
using *c2 c32 c43 c44 c45 c46 c51* **by** *blast*
qed
qed
have *c6*: (*nrev M* ⊨$_x$ *fchopstar fe*) ⟹(*M* ⊨$_y$ *rvffe* (*fchopstar fe*))
proof −
assume *a1*: (*nrev M* ⊨$_x$ *fchopstar fe*)
show (*M* ⊨$_y$ *rvffe* (*fchopstar fe*))
proof −
obtain *l* **where** *c60*: *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength M* ∧
nfinite (*nrev M*) ∧ (∀ *i*. *enat i* < *nlength l* ⟶ ((*nsubn* (*nrev M*) (*nnth l i*) (*nnth l* (*Suc i*))) ⊨$_x$ *fe*))
using *a1 c2* **by** *blast*
have *c61*: *nfinite M*
by (*simp add*: *fchopstar-d.prems*)
have *c62*: (∀ *i*. *enat i* < *nlength l* ⟶ ((*nrev* (*nsubn* (*nrev M*) (*nnth l i*) (*nnth l* (*Suc i*)))) ⊨$_y$ (*rvffe fe*)))
by (*simp add*: *c60 fchopstar-d.IH nfinite-nrev nrev-nrev nsubn-nfinite*)
have *c63*: ⋀ *i*. *enat i* < *nlength l* ⟹ (*nnth l i*) ≤ (*nnth l* (*Suc i*))
by (*simp add*: *Suc-ile-eq c60 nidx-less-eq*)
have *c64*: ⋀ *i*. *enat i* < *nlength l* ⟹ (*nnth l* (*Suc i*)) ≤ *nlength* (*nrev M*)
by (*metis c60 eSuc-enat enat-ord-simps(1) ileI1 nidx-all-le-nlast nlength-nrev*)
have *c65*: ⋀*i*. *enat i* < *nlength l* ⟹
(*nrev* (*nsubn* (*nrev M*) (*nnth l i*) (*nnth l* (*Suc i*)))) =
(*nsubn M*
((*the-enat*(*nlength* (*nrev M*))) −(*nnth l* (*Suc i*)))
((*the-enat*(*nlength* (*nrev M*))) −(*nnth l i*)))
by (*metis c60 c63 c64 enat-ord-simps(1) nrev-nrev nrev-nsubn*)
have *c360*: ∃ *x*. (*nlength M*) = (*enat x*)
by (*metis c60*)
obtain *x* **where** *c361*: (*nlength M*) = (*enat x*)
using *c360* **by** *blast*
have *c362*: ∃ *y*. (*nlength l*) = (*enat y*)
using *c60 nfinite-conv-nlength-enat* **by** *auto*
obtain *y* **where** *c363*: (*nlength l*) = (*enat y*)
using *c362* **by** *blast*
have *c364*: ⋀*j*. *j* ≤ (*nlength l*) ⟹ (*nnth* (*nrev l*) *j*) = *nnth l* (*y* − *j*)
using *nnth-nrev*[*of l*] **by** (*simp add*: *c60 c363*)

have *c365*: $\bigwedge j.\ j \leq (nlength\ l) \Longrightarrow nnth\ l\ (y - j) \leq x$
by (*metis c361 c363 c60 diff-le-self enat-ord-simps(1) nidx-all-le-nlast*)
have *c66*: $\bigwedge i.\ enat\ i < nlength\ (nrev\ l) \Longrightarrow$
$(x - (nnth\ (nrev\ l)\ i)) <$
$(x - (nnth\ (nrev\ l)\ (Suc\ i)))$
by (*metis* (*no-types*, *lifting*) *ext Suc-diff-Suc c363 c365 c60 diff-le-self diff-less-mono2 dual-order.strict-iff-order eSuc-enat enat-ord-simps(1) ileI1 less-le-not-le nidx-expand nlength-nrev nnth-nrev the-enat.simps*)

have *c67*: $((the\text{-}enat(nlength\ M)) - nnth\ (nrev\ l)\ 0) = 0$
by (*metis c60 diff-self-eq-0 nfirst-eq-nnth-zero nfirst-nrev the-enat.simps*)
have *c68*: $((the\text{-}enat(nlength\ M)) - (nlast\ (nrev\ l))) = nlength\ M$
by (*metis c60 minus-nat.diff-0 nfirst-eq-nnth-zero nlast-nrev the-enat.simps*)
have *c69*: $nridx\ (\lambda x\ y.\ ((the\text{-}enat(nlength\ M)) - x) < ((the\text{-}enat(nlength\ M)) - y))\ (nrev\ l)$
by (*simp add*: *Suc-ile-eq c361 c66 nridx-expand*)
have *c70*: $\exists\ l1.\ nlength\ l1 = nlength\ l \wedge (\forall i.\ i \leq nlength\ l1 \longrightarrow (nnth\ l1\ i) = ((the\text{-}enat(nlength\ M)) - (nnth\ (nrev\ l)\ i)))$
by (*metis nlength-nmap nlength-nrev nnth-nmap*)
obtain *l1* **where** *c42*: $nlength\ l1 = nlength\ l \wedge (\forall i.\ i \leq nlength\ l1 \longrightarrow (nnth\ l1\ i) = ((the\text{-}enat(nlength\ M)) - (nnth\ (nrev\ l)\ i)))$
using *c70* **by** *blast*
have *c71*: $nnth\ l1\ 0 = 0$
using *c42 c67 enat-defs(1)* **by** *fastforce*
have *c72*: *nfinite l1*
by (*metis c42 c60 nfinite-conv-nlength-enat*)
have *c73*: $enat\ (nlast\ l1) = nlength\ M$
by (*metis c42 c60 c68 dual-order.refl nfinite-conv-nlength-enat nlength-nrev nnth-nlast the-enat.simps*)
have *c74*: *nidx l1*
by (*simp add*: *Suc-ile-eq c361 c42 c66 nidx-expand*)
have *c75*: $\bigwedge j.\ j \leq nlength\ l \Longrightarrow$
$((the\text{-}enat(nlength\ M)) - (nnth\ (nrev\ l)\ j)) =$
$((the\text{-}enat(nlength\ M)) - (nnth\ l\ ((the\text{-}enat\ (nlength\ l)) - j)))$
by (*simp add*: *c60 nnth-nrev*)
have *c76*: $\bigwedge k.\ k \leq nlength\ l \Longrightarrow$
$((the\text{-}enat(nlength\ M)) - (nnth\ (nrev\ l)\ ((the\text{-}enat\ (nlength\ l)) - k))) =$
$((the\text{-}enat(nlength\ M)) - (nnth\ l\ k))$
by (*metis c60 nfinite-nrev nlength-nrev nnth-nrev nrev-nrev*)
have *c77*: $\bigwedge k.\ k < nlength\ l1 \Longrightarrow$
$(nsubn\ M$
$((the\text{-}enat(nlength\ (nrev\ M))) - (nnth\ l\ (Suc\ k)))$
$((the\text{-}enat(nlength\ (nrev\ M))) - (nnth\ l\ k))) =$
$(nsubn\ M$
$((the\text{-}enat(nlength\ (nrev\ M))) - (nnth\ (nrev\ l)\ ((the\text{-}enat\ (nlength\ l)) - (Suc\ k)))\ )$
$((the\text{-}enat(nlength\ (nrev\ M))) - (nnth\ (nrev\ l)\ ((the\text{-}enat\ (nlength\ l)) - k))))$
by (*simp add*: *Suc-ile-eq c42 c76*)
have *c78*: $(\forall i.\ enat\ i < nlength\ l1 \longrightarrow$

((*nsubn M* ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)) ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) (*Suc i*)))) $\models_y$ (*rvffe fe*)))
by (*metis Suc-diff-Suc Suc-ile-eq c363 c42 c60 c62 c65 diff-le-self dual-order.strict-iff-order enat-ord-simps*(*1*) *nlength-nrev nnth-nrev*
the-enat.simps)
have *c79*: (∀ *i*. *enat i* < *nlength l1* ⟶
((*nsubn M* (*nnth l1 i*) (*nnth l1* (*Suc i*))) $\models_y$ (*rvffe fe*)))
by (*simp add*: *Suc-ile-eq c42 c78*)
show *?thesis*
using *c1 c61 c71 c72 c73 c74 c79* **by** *blast*
qed
qed
show *?thesis*
using *c3 c6* **by** *blast*
qed
next
case (*ffe-or-d fe1 fe2*)
then show *?case* **by** *simp*
qed

lemma *rvpfe-sem*:
assumes *nfinite M*
shows (*M* $\models_x$ (*rvpfe pe*)) = ((*nrev M*) $\models_y$ *pe*)
using *assms*
proof (*induction pe arbitrary*: *M*)
case (*ptest-d x*)
then show *?case* **apply** *simp*
by (*metis NNil-eq-ntake-iff i0-lb nrev-NNil ntake-all*)
next
case (*pstep-d x*)
then show *?case* **apply** *simp*
using *rvpt-sem* **by** *blast*
next
case (*pechop-d pe1 pe2*)
then show *?case* **apply** *auto*
apply (*metis enat-the-enat infinity-ileE le-less ndropn-all*
not-le-imp-less nrev-ndropn nrev-ntaken ntaken-all)
by (*metis* (*no-types*, *lifting*) *diff-zero ndropn-0 ndropn-all ndropn-nlast nfinite-nrev nlength-nrev not-less nrev-ndropn nrev-nrev*
nrev-ntaken ntaken-all order-less-imp-le)
next
case (*pchopstar-d pe*)
then show *?case*
proof −
have *c1*: (*M* $\models_x$ *rvpfe* (*pchopstar pe*)) =
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength M* ∧
nfinite M ∧ (∀ *i*. *enat i* < *nlength l* ⟶ ((*nsubn M* (*nnth l i*) (*nnth l* (*Suc i*))) $\models_x$ *rvpfe pe*)))

by *simp*
have *c2*: (*nrev M* $\models_y$ *pchopstar pe*) =
(∃ *l*. *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength M* ∧
nfinite (*nrev M*) ∧ (∀ *i*. *enat i* < *nlength l* ⟶ ((*nsubn* (*nrev M*) (*nnth l i*) (*nnth l* (*Suc i*))) $\models_y$ *pe*)))
by *simp*
have *c3*: (*M* $\models_x$ *rvpfe* (*pchopstar pe*)) ⟹ (*nrev M* $\models_y$ *pchopstar pe*)
proof −
assume *a0*: (*M* $\models_x$ *rvpfe* (*pchopstar pe*))
show (*nrev M* $\models_y$ *pchopstar pe*)
proof −
obtain *l* **where** *c31*: *nnth l 0* = *0* ∧
nfinite l ∧
nidx l ∧
enat (*nlast l*) = *nlength M* ∧
nfinite M ∧ (∀ *i*. *enat i* < *nlength l* ⟶ ((*nsubn M* (*nnth l i*) (*nnth l* (*Suc i*))) $\models_x$ *rvpfe pe*))
using *a0 c1* **by** *blast*
have *c32*: *nfinite* (*nrev M*)
by (*simp add*: *pchopstar-d.prems nfinite-nrev*)
have *c33*: (∀ *i*. *enat i* < *nlength l* ⟶ ((*nrev* (*nsubn M* (*nnth l i*) (*nnth l* (*Suc i*)))) $\models_y$ *pe*))
by (*meson c31 pchopstar-d.IH nsubn-nfinite*)
have *c34*: ⋀ *j*. *j* < *nlength l* ⟹ (*nnth l j*) ≤ (*nnth l* (*Suc j*))
using *Suc-ile-eq c31 nidx-less-eq* **by** *auto*
have *c35*: ⋀*j*. *j* ≤ *nlength l* ⟹ (*nnth l j*) ≤ *nlength M*
by (*metis c31 enat-ord-simps*(*1*) *nidx-all-le-nlast*)
have *c36*: ⋀*j*. *j* < *nlength l* ⟹
(*nrev* (*nsubn M* (*nnth l j*) (*nnth l* (*Suc j*)))) =
nsubn (*nrev M*) ((*the-enat*(*nlength M*)) −(*nnth l* (*Suc j*))) ((*the-enat*(*nlength M*)) −(*nnth l j*))
by (*simp add*: *Suc-ile-eq c31 c34 c35 nrev-nsubn*)
have *c360*: ∃ *x*. (*nlength M*) = (*enat x*)
by (*metis c31*)
obtain *x* **where** *c361*: (*nlength M*) = (*enat x*)
using *c360* **by** *blast*
have *c362*: ∃ *y*. (*nlength l*) = (*enat y*)
using *c31 nfinite-conv-nlength-enat* **by** *auto*
obtain *y* **where** *c363*: (*nlength l*) = (*enat y*)
using *c362* **by** *blast*
have *c364*: ⋀*j*. *j* ≤ (*nlength l*) ⟹ (*nnth* (*nrev l*) *j*) = *nnth l* (*y* − *j*)
using *nnth-nrev*[*of l*] **by** (*simp add*: *c31 c363*)
have *c365*: ⋀*j*. *j* ≤ (*nlength l*) ⟹ *nnth l* (*y* − *j*) ≤ *x*
using *c35 c361 c363* **by** *auto*
have *c37*: ⋀*j*. *j* < *nlength* (*nrev l*) ⟹
(*x* − (*nnth* (*nrev l*) *j*)) <
(*x* − (*nnth* (*nrev l*) (*Suc j*)))
by (*metis* (*no-types*, *lifting*) *ext Suc-diff-Suc c31 c363 c365 diff-le-self diff-less-mono2 dual-order.strict-iff-order eSuc-enat enat-ord-simps*(*1*)

ileI1 less-le-not-le nidx-expand nlength-nrev nnth-nrev the-enat.simps)
have *c38*: ((*the-enat*(*nlength M*)) − *nnth* (*nrev l*) *0*) = *0*
by (*metis c31 dual-order.strict-iff-order nfirst-eq-nnth-zero*
nfirst-nrev not-gr-zero the-enat.simps zero-less-diff)
have *c39*: ((*the-enat*(*nlength M*)) − (*nlast* (*nrev l*))) = *nlength M*
by (*metis c31 minus-nat.diff-0 nfirst-eq-nnth-zero nlast-nrev*
the-enat.simps)
have *c40*: *nridx* (λ*x y*. ((*the-enat*(*nlength M*)) − *x*) < ((*the-enat*(*nlength M*)) − *y*)) (*nrev l*)
by (*simp add*: *Suc-ile-eq c361 c37 nridx-expand*)
have *c41*: ∃ *l1*. *nlength l1* = *nlength l* ∧ (∀ *i*. *i*≤ *nlength l1* ⟶ (*nnth l1 i*) = ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)))
by (*metis nlength-nmap nlength-nrev nnth-nmap*)
obtain *l1* **where** *c42*: *nlength l1* = *nlength l* ∧ (∀ *i*. *i*≤ *nlength l1* ⟶ (*nnth l1 i*) = ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)))
using *c41* **by** *blast*
have *c43*: *nnth l1 0* = *0*
using *c38 c42 enat-defs*(*1*) **by** *fastforce*
have *c44*: *nfinite l1*
by (*metis c31 c42 nfinite-conv-nlength-enat*)
have *c45*: *enat* (*nlast l1*) = *nlength M*
by (*metis c39 c42 c44 dual-order.refl nfinite-conv-nlength-enat*
nlength-nrev nnth-nlast the-enat.simps)
have *c46*: *nidx l1*
by (*simp add*: *Suc-ile-eq c361 c37 c42 nidx-expand*)
have *c47*: ⋀*j*. *j* ≤ *nlength l* ⟹
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *j*)) =
((*the-enat*(*nlength M*)) − (*nnth l* ((*the-enat* (*nlength l*))− *j*)))
by (*simp add*: *c31 nnth-nrev*)

have *c48*: ⋀*k*. *k* ≤ *nlength l* ⟹
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− *k*))) =
((*the-enat*(*nlength M*)) − (*nnth l k*))
by (*metis c31 nfinite-nrev nlength-nrev nnth-nrev*
nrev-nrev)
have *c49*: ⋀*k*. *k* < *nlength l1* ⟹
nsubn (*nrev M*) ((*the-enat*(*nlength M*)) −(*nnth l* (*Suc k*))) ((*the-enat*(*nlength M*)) −(*nnth l k*)) =
nsubn (*nrev M*) ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− (*Suc k*))))
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− *k*)))
by (*simp add*: *Suc-ile-eq c42 c48*)
have *c500*: (∀ *i*. *enat i* < *nlength l* ⟶
(*nsubn* (*nrev M*) ((*the-enat*(*nlength M*)) −(*nnth l* (*Suc i*)))
((*the-enat*(*nlength M*)) −(*nnth l i*)) ⊨$_y$ *pe*))
using *c33 c36* **by** *auto*
have *c50*: (∀ *i*. *enat i* < *nlength l1* ⟶
((*nsubn* (*nrev M*) ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)) ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) (*Suc i*)))) ⊨$_y$ *pe*))
by (*metis Suc-diff-Suc Suc-ile-eq c363 c364 c42 c500 diff-le-self enat-ord-simps*(*1*) *less-le-not-le*)
have *c51*: (∀ *i*. *enat i* < *nlength l1* ⟶
((*nsubn* (*nrev M*) (*nnth l1 i*) (*nnth l1* (*Suc i*))) ⊨$_y$ *pe*))

```
          by (simp add: Suc-ile-eq c42 c50)
     show ?thesis
     using c2 c32 c43 c44 c45 c46 c51 by blast
  qed
 qed
 have c6: (nrev M   ⊨y pchopstar pe ) ⟹(M   ⊨x rvpfe (pchopstar pe ))
   proof −
    assume a1: (nrev M   ⊨y pchopstar pe )
    show (M   ⊨x rvpfe (pchopstar pe ))
    proof −
     obtain l where c60: nnth l 0 = 0 ∧
         nfinite l ∧
         nidx l ∧
         enat (nlast l) = nlength M ∧
         nfinite (nrev M) ∧ (∀ i. enat i < nlength l ⟶ ((nsubn (nrev M) (nnth l i) (nnth l (Suc i)))   ⊨y
pe))
      using a1 c2 by blast
    have c61: nfinite M
     by (simp add: pchopstar-d.prems)
    have c62: (∀ i. enat i < nlength l ⟶ ((nrev (nsubn (nrev M) (nnth l i) (nnth l (Suc i))))   ⊨x (rvpfe
pe)))
     by (simp add: c60 pchopstar-d.IH nfinite-nrev nrev-nrev
      nsubn-nfinite)
    have c63: ⋀ i.  enat i < nlength l ⟹ (nnth l i) ≤ (nnth l (Suc i))
    by (simp add: Suc-ile-eq c60 nidx-less-eq)
    have c64: ⋀ i.  enat i < nlength l ⟹ (nnth l (Suc i)) ≤ nlength (nrev M)
    by (metis c60 eSuc-enat enat-ord-simps(1) ileI1
      nidx-all-le-nlast nlength-nrev)
    have c65: ⋀i. enat i < nlength l ⟹
              (nrev (nsubn (nrev M) (nnth l i) (nnth l (Suc i)))) =
             (nsubn M
                  ((the-enat(nlength (nrev M))) −(nnth l (Suc i)))
                  ((the-enat(nlength (nrev M))) −(nnth l i)))
       by (metis c60 c63 c64 enat-ord-simps(1) nrev-nrev
                    nrev-nsubn)
    have c360: ∃ x. (nlength M) = (enat x)
    by (metis c60)
    obtain x where  c361: (nlength M) = (enat x)
     using c360 by blast
    have c362: ∃ y. (nlength l) = (enat y)
    using c60 nfinite-conv-nlength-enat by auto
    obtain y where  c363: (nlength l) = (enat y)
      using c362 by blast
    have c364: ⋀j. j ≤ (nlength l) ⟹ (nnth (nrev l) j) = nnth l (y − j)
       using nnth-nrev[of l] by (simp add: c60 c363)
    have c365: ⋀j. j ≤ (nlength l) ⟹ nnth l (y − j) ≤ x
    by (metis c361 c363 c60 diff-le-self enat-ord-simps(1) nidx-all-le-nlast)
    have c66: ⋀i. enat i < nlength (nrev l) ⟹
                (x −(nnth (nrev l) i)) <
                (x −(nnth (nrev l) (Suc i)))
```

by (*metis* (*no-types*, *lifting*) *ext Suc-diff-Suc c363 c365 c60 diff-le-self diff-less-mono2 dual-order.strict-iff-order eSuc-enat enat-ord-simps*(*1*)
ileI1 less-le-not-le nidx-expand nlength-nrev nnth-nrev the-enat.simps)
have *c67*: ((*the-enat*(*nlength M*)) − *nnth* (*nrev l*) *0*) = *0*
by (*metis c60 diff-self-eq-0 nfirst-eq-nnth-zero nfirst-nrev*
the-enat.simps)
have *c68*: ((*the-enat*(*nlength M*)) − (*nlast* (*nrev l*))) = *nlength M*
by (*metis c60 minus-nat.diff-0 nfirst-eq-nnth-zero nlast-nrev*
the-enat.simps)
have *c69*: *nridx* (λ*x y*. ((*the-enat*(*nlength M*)) − *x*) < ((*the-enat*(*nlength M*)) − *y*)) (*nrev l*)
using *Suc-ile-eq c66 nridx-expand c361* **by** *force*
have *c70*: ∃ *l1*. *nlength l1* = *nlength l* ∧ (∀ *i*. *i*≤ *nlength l1* ⟶ (*nnth l1 i*) = ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)))
by (*metis nlength-nmap nlength-nrev nnth-nmap*)
obtain *l1* **where** *c42*: *nlength l1* = *nlength l* ∧ (∀ *i*. *i*≤ *nlength l1* ⟶ (*nnth l1 i*) = ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)))
using *c70* **by** *blast*
have *c71*: *nnth l1 0* = *0*
using *c42 c67 enat-defs*(*1*) **by** *fastforce*
have *c72*: *nfinite l1*
by (*metis c42 c60 nfinite-conv-nlength-enat*)
have *c73*: *enat* (*nlast l1*) = *nlength M*
by (*metis c42 c60 c68 dual-order.refl nfinite-conv-nlength-enat*
nlength-nrev nnth-nlast the-enat.simps)
have *c74*: *nidx l1*
using *Suc-ile-eq c42 c66 nidx-expand*
using *c361* **by** *auto*
have *c75*: ⋀*j*. *j* ≤ *nlength l* ⟹
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *j*)) =
((*the-enat*(*nlength M*)) − (*nnth l* ((*the-enat* (*nlength l*))− *j*)))
by (*simp add*: *c60 nnth-nrev*)
have *c76*: ⋀*k*. *k* ≤ *nlength l* ⟹
((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− *k*))) =
((*the-enat*(*nlength M*)) − (*nnth l k*))
by (*metis c60 nfinite-nrev nlength-nrev nnth-nrev*
nrev-nrev)
have *c77*: ⋀*k*. *k* < *nlength l1* ⟹
(*nsubn M*
((*the-enat*(*nlength* (*nrev M*))) −(*nnth l* (*Suc k*)))
((*the-enat*(*nlength* (*nrev M*))) −(*nnth l k*))) =
(*nsubn M*
((*the-enat*(*nlength* (*nrev M*))) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− (*Suc k*))))
((*the-enat*(*nlength* (*nrev M*))) − (*nnth* (*nrev l*) ((*the-enat* (*nlength l*))− *k*))))
by (*simp add*: *Suc-ile-eq c42 c76*)
have *c360*: ∃ *x*. (*nlength M*) = (*enat x*)
by (*metis c60*)
obtain *x* **where** *c361*: (*nlength M*) = (*enat x*)
using *c360* **by** *blast*
have *c362*: ∃ *y*. (*nlength l*) = (*enat y*)
using *c60 nfinite-conv-nlength-enat* **by** *auto*

obtain *y* **where** *c363*: (*nlength l*) = (*enat y*)
using *c362* **by** *blast*
have *c364*: $\bigwedge j.\ j \leq$ (*nlength l*) $\Longrightarrow$ (*nnth* (*nrev l*) *j*) = *nnth l* (*y* − *j*)
using *nnth-nrev*[*of l*] **by** (*simp add*: *c60 c363*)
have *c365*: $\bigwedge j.\ j \leq$ (*nlength l*) $\Longrightarrow$ *nnth l* (*y* − *j*) ≤ *x*
by (*metis c361 c363 c60 diff-le-self enat-ord-simps*(*1*) *nidx-all-le-nlast*)
have *c66*: $\bigwedge i.$ *enat i* < *nlength* (*nrev l*) $\Longrightarrow$
(*x* −(*nnth* (*nrev l*) *i*)) <
(*x* −(*nnth* (*nrev l*) (*Suc i*)))
by (*metis* (*no-types*, *lifting*) *ext Suc-diff-Suc c363 c365 c60 diff-le-self diff-less-mono2 dual-order.strict-iff-order eSuc-enat enat-ord-simps*(*1*)
ileI1 less-le-not-le nidx-expand nlength-nrev nnth-nrev the-enat.simps)

have *c78*: (∀ *i*. *enat i* < *nlength l1* ⟶
((*nsubn M* ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) *i*)) ((*the-enat*(*nlength M*)) − (*nnth* (*nrev l*) (*Suc i*)))) $\models_x$ (*rvpfe pe*)))
by (*metis Suc-diff-Suc Suc-ile-eq c363 c42 c60 c62 c65 diff-le-self dual-order.strict-iff-order enat-ord-simps*(*1*) *nlength-nrev nnth-nrev*
the-enat.simps)
have *c79*: (∀ *i*. *enat i* < *nlength l1* ⟶
((*nsubn M* (*nnth l1 i*) (*nnth l1* (*Suc i*))) $\models_x$ (*rvpfe pe*)))
by (*simp add*: *Suc-ile-eq c42 c78*)
show *?thesis*
using *c1 c61 c71 c72 c73 c74 c79* **by** *blast*
qed
qed
show *?thesis*
using *c3 c6* **by** *blast*
qed
next
case (*pfe-or-d pe1 pe2*)
then show *?case* **by** *simp*
qed

lemma *rv-sem-help*:
fixes *l* :: *nat*
fixes *k* :: *nat*
fixes *i* :: *nat*
assumes *i* ≤ *l*
k ≤ *l*
shows (∀ *n*. *l* − *i* < *n* ∧ *n* < *k* ⟶ *P* (*l* −*n*)) = (∀ *n*. (*l* − *k*) < *n* ∧ *n* < *i* ⟶ *P n*)
using *assms diff-less-mono2* **apply** *auto*
by (*metis diff-diff-cancel dual-order.strict-trans1 le-diff-iff*′ *linorder-not-less order-less-imp-not-less*)

lemma *rv-sem-help-alt*:
fixes *l* :: *nat*
fixes *k* :: *nat*
fixes *i* :: *nat*
assumes *i* ≤ *l*
k ≤ *l*

shows $(\forall n.\ i < n \wedge n < k \longrightarrow P\ (l - n)) =$
$(\forall n.\ (l - k) < n \wedge n < l - i \longrightarrow P\ n)$
using *assms rv-sem-help*[*of l−i l k P*]
by *simp*

lemma *rv-sem-help-alt-1*:
fixes l :: *nat*
fixes k :: *nat*
fixes i :: *nat*
assumes $i \leq l$
$k \leq l$
shows $(\forall n.\ k < n \wedge n < l - i \longrightarrow P\ (\ l - n)\ ) = (\forall n.\ i < n \wedge n < l\ - k \longrightarrow P\ n)$
using *assms*
rv-sem-help[*of l−k l l−i P*]
by *simp*

lemma *rv-sem*:
assumes *nfinite M*
i≤ *nlength M*
shows $(\ M\ i \models (rv\ f)) = (\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i) \models f)$
using *assms*
proof (*induction f arbitrary*: *M i*)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case* **apply** *simp*
by (*metis nfinite-nrev nlength-nrev nnth-nrev*
nrev-nrev)
next
next
case (*iimp-d f1 f2*)
then show *?case* **by** *simp*
next
case (*chop-d f1 f2*)
then show *?case*
proof −
have *c1*: $(M\ i\ \models rv\ (\ f1\ chop\ f2\ )) =$
$(\exists k{\leq}i.\ (M\ k\ \models rv\ f2) \wedge (nsubn\ M\ k\ i\ \models_y rvffe\ f1))$
by *simp*
have *c2*: $((nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ \models\ f1\ chop\ f2\ ) =$
$(\exists k{\geq}the\text{-}enat\ (nlength\ M) - i.$
$enat\ k \leq nlength\ M \wedge (nsubn\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ k\ \models_x f1) \wedge ((nrev\ M)\ k\ \models f2))$
by *simp*
have *c3*: $(\exists k{\leq}i.\ (M\ k\ \models rv\ f2) \wedge (nsubn\ M\ k\ i\ \models_y rvffe\ f1)) \Longrightarrow$
$(\exists k{\geq}the\text{-}enat\ (nlength\ M) - i.$
$enat\ k \leq nlength\ M \wedge (nsubn\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ k\ \models_x f1) \wedge ((nrev\ M)\ k\ \models f2))$
proof −

assume *ca0*: $(\exists\, k{\leq}i.\ (M\ k\ \models rv\ f2) \wedge (nsubn\ M\ k\ i\ \models_y rvffe\ f1))$
show $(\exists\, k{\geq}$*the-enat* $(nlength\ M) - i.$
$enat\ k \leq nlength\ M \wedge (nsubn\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ k\ \models_x f1) \wedge ((nrev\ M)\ k\ \models f2))$
proof −
obtain k **where** *c4*: $k{\leq}i \wedge (M\ k\ \models rv\ f2) \wedge (nsubn\ M\ k\ i\ \models_y rvffe\ f1)$
using *ca0* **by** *blast*
have *c5*: $((nrev\ M)\ (the\text{-}enat\ (nlength\ M) - k)\ \models\ f2)$
by (*metis c4 chop-d.IH chop-d.prems*(*1*,*2*) *enat-ord-simps*(*1*)
order.trans)
have *c6*: *nfinite* (*nsubn M k i*)
by (*simp add*: *nsubn-nfinite*)
have *c7*: $enat\ (i - k) \leq nlength\ (ndropn\ k\ M)$
by (*metis chop-d.prems*(*1*,*2*) *diff-le-mono*
enat-ord-simps(*1*) *idiff-enat-enat ndropn-nlength*
nfinite-conv-nlength-enat)
have *c8*: $(nsubn\ M\ k\ i\ \models_y rvffe\ f1) =$
$((nrev\ (nsubn\ M\ k\ i)) \models_x f1)$
by (*simp add*: *c6 rvffe-sem*)
have *c9*: $(nrev\ (nsubn\ M\ k\ i)) = nsubn\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ (the\text{-}enat\ (nlength\ M) - k)$
by (*simp add*: *c4 chop-d.prems*(*1*,*2*) *nrev-nsubn*)
show *?thesis*
by (*metis c4 c5 c8 c9 chop-d.prems*(*1*) *diff-le-mono2 diff-le-self*
enat-ord-simps(*1*) *nfinite-conv-nlength-enat*
the-enat.simps)
qed
qed
have *c11*: $(\exists\, k{\geq}$*the-enat* $(nlength\ M) - i.$
$enat\ k \leq nlength\ M \wedge (nsubn\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ k\ \models_x f1) \wedge ((nrev\ M)\ k\ \models f2))$
$\Longrightarrow$
$(\exists\, k{\leq}i.\ (M\ k\ \models rv\ f2) \wedge (nsubn\ M\ k\ i\ \models_y rvffe\ f1))$
proof −
assume *ac1*: $(\exists\, k{\geq}$*the-enat* $(nlength\ M) - i.$
$enat\ k \leq nlength\ M \wedge (nsubn\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ k\ \models_x f1) \wedge ((nrev\ M)\ k\ \models f2))$
show $(\exists\, k{\leq}i.\ (M\ k\ \models rv\ f2) \wedge (nsubn\ M\ k\ i\ \models_y rvffe\ f1))$
proof −
obtain k **where** *c12*: $k{\geq}$*the-enat* $(nlength\ M) - i \wedge$
$enat\ k \leq nlength\ M \wedge (nsubn\ (nrev\ M)\ (the\text{-}enat\ (nlength\ M) - i)\ k\ \models_x f1) \wedge ((nrev\ M)\ k\ \models f2)$
using *ac1* **by** *blast*
have *c13*: $(M\ (the\text{-}enat\ (nlength\ M) - k)\ \models rv\ f2)$
by (*metis add-diff-cancel-left' c12 chop-d.IH chop-d.prems*(*1*)
diff-add diff-le-self enat-ord-simps(*1*)
nfinite-conv-nlength-enat the-enat.simps)
have *c14*: $(nsubn\ M\ (the\text{-}enat\ (nlength\ M) - k)\ i\ \models_y rvffe\ f1)$
by (*metis c12 chop-d.prems*(*1*,*2*) *diff-diff-cancel diff-le-self*
enat-ord-simps(*1*) *le-diff-iff' nfinite-conv-nlength-enat*
nrev-nsubn nsubn-nfinite rvffe-sem the-enat.simps)
show *?thesis*
by (*metis c12 c13 c14 diff-is-0-eq diff-right-commute*)
qed

```
  qed
  show ?thesis using c11 c3 by auto
 qed
case (pchop-d f1 f2)
 then show ?case
  proof −
   have p1: (M i ⊨ rv ( f1 pchop f2 )) =
          (∃k≥i. enat k ≤ nlength M ∧ (nsubn M i k ⊨x rvpfe f2) ∧ (M k ⊨ rv f1))
   by simp
   have p2: ((nrev M) (the-enat (nlength M) − i) ⊨ f1 pchop f2 ) =
             (∃k≤the-enat (nlength M) − i. ((nrev M) k ⊨ f1) ∧ (nsubn (nrev M) k (the-enat (nlength
M) − i) ⊨y f2))
    by simp
   have p3: (∃k≥i. enat k ≤ nlength M ∧ (nsubn M i k ⊨x rvpfe f2) ∧ (M k ⊨ rv f1)) ⟹
          (∃k≤the-enat (nlength M) − i. ((nrev M) k ⊨ f1) ∧ (nsubn (nrev M) k (the-enat (nlength M)
− i) ⊨y f2))
   proof −
    assume pa0: (∃k≥i. enat k ≤ nlength M ∧ (nsubn M i k ⊨x rvpfe f2) ∧ (M k ⊨ rv f1))
    show (∃k≤the-enat (nlength M) − i. ((nrev M) k ⊨ f1) ∧ (nsubn (nrev M) k (the-enat (nlength M)
− i) ⊨y f2))
    proof −
     obtain k where p30: k≥i ∧ enat k ≤ nlength M ∧ (nsubn M i k ⊨x rvpfe f2) ∧ (M k ⊨ rv f1)
      using pa0 by blast
     have p31: ((nrev M) (the-enat (nlength M) − k) ⊨ f1)
     using p30 pchop-d.IH pchop-d.prems(1) by blast
     have p32: (nsubn (nrev M) (the-enat (nlength M) − k) (the-enat (nlength M) − i) ⊨y f2)
     by (metis enat-ord-simps(1) nrev-nsubn nsubn-nfinite p30
       pchop-d.prems(1) rvpfe-sem)
     show ?thesis
     using diff-le-mono2 p30 p31 p32 by blast
   qed
  qed
  have p4: (∃k≤the-enat (nlength M) − i. ((nrev M) k ⊨ f1) ∧ (nsubn (nrev M) k (the-enat (nlength M)
− i) ⊨y f2))
          ⟹
          (∃k≥i. enat k ≤ nlength M ∧ (nsubn M i k ⊨x rvpfe f2) ∧ (M k ⊨ rv f1))
  proof −
   assume pa1: (∃k≤the-enat (nlength M) − i. ((nrev M) k ⊨ f1) ∧ (nsubn (nrev M) k (the-enat (nlength
M) − i) ⊨y f2))
   show (∃k≥i. enat k ≤ nlength M ∧ (nsubn M i k ⊨x rvpfe f2) ∧ (M k ⊨ rv f1))
   proof −
    obtain k where p40: k≤the-enat (nlength M) − i ∧ ((nrev M) k ⊨ f1) ∧ (nsubn (nrev M) k (the-enat
(nlength M) − i) ⊨y f2)
    using pa1 by blast
    have p410: k ≤ (the-enat (nlength M))
     using diff-le-self le-trans p40 by blast
    have p41: (M (the-enat (nlength M) − k) ⊨ rv f1)
     by (metis diff-diff-cancel diff-le-self enat-ord-simps(1) enat-the-enat infinity-ileE linorder-linear p40
p410 pchop-d.IH
       pchop-d.prems(1))
```

have *p420*: $i \leq (\textit{the-enat}\ (\textit{nlength}\ M) - k)$
by (*metis diff-diff-cancel diff-le-mono2 enat.simps(3) enat-ord-simps(1) enat-the-enat nfinite-conv-nlength-enat p40*
pchop-d.prems(1,2))
have *p42*: $(\textit{nsubn}\ M\ i\ (\textit{the-enat}\ (\textit{nlength}\ M) - k) \models_x \textit{rvpfe}\ f2)$
by (*metis diff-diff-cancel diff-le-self enat-ord-simps(1) nfinite-conv-nlength-enat nrev-nsubn nsubn-nfinite p40 p410 p420*
pchop-d.prems(1) rvpfe-sem the-enat.simps)
show *?thesis*
by (*metis diff-diff-cancel diff-le-mono2 diff-le-self*
enat.simps(3) enat-ord-simps(1) enat-the-enat
nfinite-conv-nlength-enat p40 p41 p42
pchop-d.prems(1,2))
qed
qed
show *?thesis*
using *p1 p2 p3 p4* **by** *blast*
qed
case (*until-d f1 f2*)
then show *?case*
proof −
have *u1*: $(M\ i \models \textit{rv}\ (\ f1\ \textit{until}\ f2\ )) =$
$(\exists\, k{\leq}i.\ (M\ k \models \textit{rv}\ f2) \wedge (\forall\, n.\ k < n \wedge n < i \longrightarrow (M\ n \models \textit{rv}\ f1)))$
by *simp*
have *u2*: $((\textit{nrev}\ M)\ (\textit{the-enat}\ (\textit{nlength}\ M) - i) \models f1\ \textit{until}\ f2\ ) =$
$(\exists\, k{\geq}\textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - i.\ \textit{enat}\ k \leq \textit{nlength}\ (\textit{nrev}\ M) \wedge ((\textit{nrev}\ M)\ k \models f2) \wedge$
$(\forall\, n.\ \textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - i < n \wedge n < k \longrightarrow ((\textit{nrev}\ M)\ n \models f1)))$
by *simp*
have *u3*: $(\exists\, k{\leq}i.\ (M\ k \models \textit{rv}\ f2) \wedge (\forall\, n.\ k < n \wedge n < i \longrightarrow (M\ n \models \textit{rv}\ f1))) \Longrightarrow$
$(\exists\, k{\geq}\textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - i.\ \textit{enat}\ k \leq \textit{nlength}\ (\textit{nrev}\ M) \wedge$
$((\textit{nrev}\ M)\ k \models f2) \wedge (\forall\, n.\ \textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - i < n \wedge n < k \longrightarrow ((\textit{nrev}\ M)\ n \models f1)))$
proof −
assume *ua0*: $(\exists\, k{\leq}i.\ (M\ k \models \textit{rv}\ f2) \wedge (\forall\, n.\ k < n \wedge n < i \longrightarrow (M\ n \models \textit{rv}\ f1)))$
show $(\exists\, k{\geq}\textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - i.\ \textit{enat}\ k \leq \textit{nlength}\ (\textit{nrev}\ M) \wedge$
$((\textit{nrev}\ M)\ k \models f2) \wedge (\forall\, n.\ \textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - i < n \wedge n < k \longrightarrow ((\textit{nrev}\ M)\ n \models f1)))$
proof −
obtain *k* **where** *u100*: $k{\leq}i \wedge (M\ k \models \textit{rv}\ f2) \wedge (\forall\, n.\ k < n \wedge n < i \longrightarrow (M\ n \models \textit{rv}\ f1))$
using *ua0* **by** *blast*
have *u101*: $((\textit{nrev}\ M)\ (\textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - k) \models f2)$
by (*metis enat-ord-simps(1) nlength-nrev order-trans u100 until-d.IH(2) until-d.prems(1,2)*)
have *u102*: $(\textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - k) \geq \textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)) - i$
by (*simp add*: *diff-le-mono2 u100*)
have *u1030*: $k \leq (\textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M)))$
by (*metis dual-order.trans enat-ord-simps(1) nfinite-conv-nlength-enat nlength-nrev the-enat.simps u100 until-d.prems(1,2)*)
have *u103*: $(\forall\, n.\ k < n \wedge n < i \longrightarrow ((\textit{nrev}\ M)\ (\textit{the-enat}(\textit{nlength}\ (\textit{nrev}\ M)) - n) \models f1))$
by (*metis enat-ord-simps(2) less-le-not-le nlength-nrev order-less-le-subst2 u100 until-d.IH(1) until-d.prems(1,2)*)
have *u1040*: $i \leq \textit{the-enat}\ (\textit{nlength}\ (\textit{nrev}\ M))$
using *nfinite-nlength-enat until-d.prems(1,2)* **by** *fastforce*

have *u104*: ($\forall$ *n. the-enat* (*nlength* (*nrev M*)) − *i* < *n* ∧ *n* < (*the-enat* (*nlength* (*nrev M*)) − *k*) ⟶ ((*nrev M*) *n* ⊨ *f1*))
using *u103 rv-sem-help*[*of the-enat* (*nlength* (*nrev M*))−*k the-enat* (*nlength* (*nrev M*)) *i* λ*n*. ((*nrev M*) *n* ⊨ *f1*)]
u1030 **by** (*metis diff-diff-cancel diff-le-self u1040*)
show *?thesis*
by (*metis diff-le-self enat-ord-simps*(*1,3*) *enat-the-enat*
u101 u102 u104)
qed
qed
have *u4*: ($\exists$ *k*≥*the-enat* (*nlength* (*nrev M*)) − *i*. *enat k* ≤ *nlength* (*nrev M*) ∧ ((*nrev M*) *k* ⊨ *f2*) ∧
($\forall$ *n*. *the-enat* (*nlength* (*nrev M*)) − *i* < *n* ∧ *n* < *k* ⟶ ((*nrev M*) *n* ⊨ *f1*))) ⟹
($\exists$ *k*≤*i*. (*M k* ⊨ *rv f2*) ∧ ($\forall$ *n*. *k* < *n* ∧ *n* < *i* ⟶ (*M n* ⊨ *rv f1*)))
proof−
assume *ua0*: ($\exists$ *k*≥*the-enat* (*nlength* (*nrev M*)) − *i*. *enat k* ≤ *nlength* (*nrev M*) ∧ ((*nrev M*) *k* ⊨ *f2*) ∧
($\forall$ *n*. *the-enat* (*nlength* (*nrev M*)) − *i* < *n* ∧ *n* < *k* ⟶ ((*nrev M*) *n* ⊨ *f1*)))
show ($\exists$ *k*≤*i*. (*M k* ⊨ *rv f2*) ∧ ($\forall$ *n*. *k* < *n* ∧ *n* < *i* ⟶ (*M n* ⊨ *rv f1*)))
proof −
obtain *k* **where** *u200*: *k*≥*the-enat* (*nlength* (*nrev M*)) − *i* ∧ *enat k* ≤ *nlength* (*nrev M*) ∧
((*nrev M*) *k* ⊨ *f2*) ∧ ($\forall$ *n*. *the-enat* (*nlength* (*nrev M*)) − *i* < *n* ∧ *n* < *k* ⟶ ((*nrev M*) *n* ⊨ *f1*))
using *ua0* **by** *blast*

have *u201*: (*M* (*the-enat*(*nlength M*) − *k*) ⊨ *rv f2*)
by (*metis diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *nfinite-conv-nlength-enat nlength-nrev the-enat.simps u200*
until-d.IH(*2*) *until-d.prems*(*1*))
have *u202*: (*the-enat*(*nlength* (*M*)) − *k*) ≤ *i*
by (*metis diff-diff-cancel diff-le-mono2 diff-le-self dual-order.trans linorder-le-cases nlength-nrev u200*)
have *u2020*: ($\exists$ *l* . *nlength M* = (*enat l*))
using *nfinite-conv-nlength-enat until-d.prems*(*1*) **by** *auto*
obtain *l* **where** *u2021*: *nlength M* = (*enat l*)
using *u2020* **by** *blast*
have *u203*: ($\forall$ *n*. *l* − *i* < *n* ∧ *n* < *k* ⟶ (*M* (*l* −*n*) ⊨ *rv f1*))
using *u200*
by (*simp add*: *u2021 until-d.IH*(*1*) *until-d.prems*(*1*))
have *u2022*: *k* ≤ *l*
using *u200 u2021* **by** *fastforce*
have *u2023*: *i* ≤ *l*
using *u2021 until-d.prems*(*2*) **by** *auto*
have *u204*: ($\forall$ *n*. (*l* − *k*) < *n* ∧ *n* < *i* ⟶ (*M n* ⊨ *rv f1*))
using *u203 u200 rv-sem-help*[*of* - - -λ*n*. (*M n* ⊨ *rv f1*)]
by (*simp add*: *u2022 u2023*)
show *?thesis*
using *u201 u202 u2021 u204* **by** *auto*
qed
qed
show *?thesis*
using *u1 u2 u3 u4* **by** *argo*
qed
next

case (*since-d f1 f2*)
then show *?case*
proof −
have *s1*: (*M i* ⊨ *rv* (*f1 since f2*)) =
(∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ (*M k* ⊨ *rv f2*) ∧ (∀ *n*. *i* < *n* ∧ *n* < *k* ⟶ (*M n* ⊨ *rv f1*)))
by *simp*
have *s2*: ((*nrev M*) (*the-enat* (*nlength M*) − *i*) ⊨ *f1 since f2*) =
(∃ *k*≤*the-enat* (*nlength* (*nrev M*)) − *i*. ((*nrev M*) *k* ⊨ *f2*) ∧
(∀ *n*. *k* < *n* ∧ *n* < *the-enat* (*nlength* (*nrev M*)) − *i* ⟶ ((*nrev M*) *n* ⊨ *f1*)))
by *simp*
have *s3*: (∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ (*M k* ⊨ *rv f2*) ∧ (∀ *n*. *i* < *n* ∧ *n* < *k* ⟶ (*M n* ⊨ *rv f1*))) ⟹
(∃ *k*≤*the-enat* (*nlength* (*nrev M*)) − *i*. ((*nrev M*) *k* ⊨ *f2*) ∧
(∀ *n*. *k* < *n* ∧ *n* < *the-enat* (*nlength* (*nrev M*)) − *i* ⟶ ((*nrev M*) *n* ⊨ *f1*)))
proof −
assume *sa0*: (∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ (*M k* ⊨ *rv f2*) ∧ (∀ *n*. *i* < *n* ∧ *n* < *k* ⟶ (*M n* ⊨ *rv f1*)))
show (∃ *k*≤*the-enat* (*nlength* (*nrev M*)) − *i*. ((*nrev M*) *k* ⊨ *f2*) ∧
(∀ *n*. *k* < *n* ∧ *n* < *the-enat* (*nlength* (*nrev M*)) − *i* ⟶ ((*nrev M*) *n* ⊨ *f1*)))
proof −
obtain *k* **where** *s100*: *k*≥*i*∧ *enat k* ≤ *nlength M* ∧ (*M k* ⊨ *rv f2*) ∧ (∀ *n*. *i* < *n* ∧ *n* < *k* ⟶ (*M n* ⊨ *rv f1*))
using *sa0* **by** *blast*
have *s101*: ((*nrev M*) (*the-enat*(*nlength M*) − *k*) ⊨ *f2*)
using *s100 since-d.IH*(*2*) *since-d.prems*(*1*) **by** *blast*
have *s102*: (*the-enat*(*nlength* (*nrev M*)) − *k*) ≤ *the-enat* (*nlength* (*nrev M*)) − *i*
by (*simp add*: *diff-le-mono2 s100*)
obtain *l* **where** *s1020*: (*enat l*) = *nlength M*
by (*metis nfinite-conv-nlength-enat since-d.prems*(*1*))
have *s103*: (∀ *n*. *i* < *n* ∧ *n* < *k* ⟶ ((*nrev M*) (*the-enat*(*nlength M*) − *n*) ⊨ *f1*))
by (*meson dual-order.trans enat-ord-simps*(*1*) *order.order-iff-strict s100 since-d.IH*(*1*) *since-d.prems*(*1*))

have *s104*: (∀ *n*. (*the-enat*(*nlength* (*nrev M*)) − *k*) < *n* ∧ *n* < *the-enat* (*nlength* (*nrev M*)) − *i* ⟶ ((*nrev M*) *n* ⊨ *f1*))
using *s103 rv-sem-help-alt*[*of* - - - λ*n*. ((*nrev M*) *n* ⊨ *f1*)]
by (*metis enat-ord-simps*(*1*) *nlength-nrev s100 s1020 since-d.prems*(*2*) *the-enat.simps*)
show *?thesis*
using *s101 s102 s104* **by** *auto*
qed
qed
have *s4*: (∃ *k*≤*the-enat* (*nlength* (*nrev M*)) − *i*. ((*nrev M*) *k* ⊨ *f2*) ∧
(∀ *n*. *k* < *n* ∧ *n* < *the-enat* (*nlength* (*nrev M*)) − *i* ⟶ ((*nrev M*) *n* ⊨ *f1*))) ⟹
(∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ (*M k* ⊨ *rv f2*) ∧ (∀ *n*. *i* < *n* ∧ *n* < *k* ⟶ (*M n* ⊨ *rv f1*)))
proof −
assume *sa0*: (∃ *k*≤*the-enat* (*nlength* (*nrev M*)) − *i*. ((*nrev M*) *k* ⊨ *f2*) ∧
(∀ *n*. *k* < *n* ∧ *n* < *the-enat* (*nlength* (*nrev M*)) − *i* ⟶ ((*nrev M*) *n* ⊨ *f1*)))
show (∃ *k*≥*i*. *enat k* ≤ *nlength M* ∧ (*M k* ⊨ *rv f2*) ∧ (∀ *n*. *i* < *n* ∧ *n* < *k* ⟶ (*M n* ⊨ *rv f1*)))
proof −
obtain *k* **where** *s200*: *k*≤*the-enat* (*nlength* (*nrev M*)) − *i*∧ ((*nrev M*) *k* ⊨ *f2*) ∧
(∀ *n*. *k* < *n* ∧ *n* < *the-enat* (*nlength* (*nrev M*)) − *i* ⟶ ((*nrev M*) *n* ⊨ *f1*))
using *sa0* **by** *blast*
have *s201*: (*M* (*the-enat*(*nlength M*) − *k*) ⊨ *rv f2*)

using *nfinite-nlength-enat s200 since-d.IH(2) since-d.prems(1)* **by** *fastforce*
have *s202*: (*the-enat*(*nlength M*) − *k*) ≥ *i*
by (*metis diff-diff-cancel diff-le-mono2 dual-order.irrefl dual-order.strict-trans1 enat-ord-simps(2) nfinite-conv-nlength-enat*
nlength-nrev not-le-imp-less s200 since-d.prems(1,2) the-enat.simps)

have *s203*: (∀ *n*. *k* < *n* ∧ *n* < *the-enat* (*nlength M*) − *i* ⟶ (*M* (*the-enat*(*nlength M*) −*n*) ⊨ *rv f1*))
using *nfinite-nlength-enat s200 since-d.IH(1) since-d.prems(1)* **by** *fastforce*

have *s204*: (∀ *n*. *i* < *n* ∧ *n* < (*the-enat*(*nlength M*) − *k*) ⟶ (*M n* ⊨ *rv f1*))
using *s203 rv-sem-help-alt-1*[*of - - -* λ*n*. (*M n* ⊨ *rv f1*)]
by *auto*
show *?thesis*
by (*metis diff-le-self enat.simps(3) enat-ord-simps(1) enat-the-enat nfinite-conv-nlength-enat s201 s202 s204*
since-d.prems(1))
qed
qed
show *?thesis*
using *s1 s2 s3 s4* **by** *argo*
qed
qed

lemma *rvpt-rvft*:
rvpt (*rvft ft*) = *ft*
proof (*induction ft*)
case (*fform x*)
then show *?case* **by** *simp*
next
case (*ftimp-d ft1 ft2*)
then show *?case* **by** *simp*
next
case (*next-d x*)
then show *?case* **by** *simp*
qed

lemma *rvft-rvpt*:
rvft (*rvpt pt*) = *pt*
proof (*induction pt*)
case (*pform x*)
then show *?case* **by** *simp*
next
case (*ptimp-d pt1 pt2*)
then show *?case* **by** *simp*
next
case (*prev-d x*)
then show *?case* **by** *simp*
qed

lemma *rvpfe-rvffe*:

rvpfe (*rvffe fe*) = *fe*
proof (*induction fe*)
case (*ftest-d x*)
then show *?case* **by** *simp*
next
case (*fstep-d x*)
then show *?case* **by** (*simp add*: *rvpt-rvft*)
next
case (*fechop-d fe1 fe2*)
then show *?case* **by** *simp*
next
case (*fchopstar-d fe*)
then show *?case* **by** *simp*
next
case (*ffe-or-d fe1 fe2*)
then show *?case* **by** *simp*
qed

lemma *rvffe-rvpfe*:
rvffe (*rvpfe pe*) = *pe*
proof (*induction pe*)
case (*ptest-d x*)
then show *?case* **by** *simp*
next
case (*pstep-d x*)
then show *?case* **by** (*simp add*: *rvft-rvpt*)
next
case (*pechop-d pe1 pe2*)
then show *?case* **by** *simp*
next
case (*pchopstar-d pe*)
then show *?case* **by** *simp*
next
case (*pfe-or-d pe1 pe2*)
then show *?case* **by** *simp*
qed

lemma *rv-rv*:
rv (*rv f*) = *f*
proof (*induction f*)
case *false-d*
then show *?case* **by** *simp*
next
case (*atom-d x*)
then show *?case* **by** *simp*
next
case (*iimp-d f1 f2*)
then show *?case* **by** *simp*
next
case (*chop-d x1 f*)

then show *?case* **using** *rvpfe-rvffe* **by** *auto*
next
case (*pchop-d f x2a*)
then show *?case* **by** (*simp add*: *rvffe-rvpfe*)
next
case (*until-d f1 f2*)
then show *?case* **by** *simp*
next
case (*since-d f1 f2*)
then show *?case* **by** *simp*
qed

lemma *rv-rv-sem*:
assumes *nfinite M*
$i \leq$ *nlength M*
shows $(M\ i \models (rv(rv\ f))) = (M\ i \models f)$
using *assms*
by (*simp add*: *rv-rv*)

lemma *fin-valid-rv*:
$(\vdash_f (rv\ f)) = (\vdash_f f)$
unfolding *valid-fin-def*
apply *auto*
apply (*metis diff-diff-cancel diff-le-self enat-ord-simps*(*1*)
nfinite-conv-nlength-enat nlength-nrev nrev-nrev rv-sem
the-enat.simps)
by (*metis diff-le-self enat-ord-simps*(*1*) *nfinite-nlength-enat nfinite-nrev nlength-nrev rv-sem the-enat.simps*)

end

theory *Semantic-View*
imports *NELList-Extras*

begin

This theory formalises the classification of properties of Manna and Pnueli [7, 8]. The formalisation is at semantic level, i.e., in terms of sets of sequences of states.

7.7 Definitions

definition *nprefix* (**infix** $\prec$ *65*)
where $nprefix\ \sigma 1\ \sigma 2 = (\exists\ \sigma 3.\ nfinite\ \sigma 1 \wedge \sigma 2 = nappend\ \sigma 1\ \sigma 3)$

definition *nprefix-eq* (**infix** $\preceq$ *65*)

where *nprefix-eq* $\sigma 1\ \sigma 2 = (\ \sigma 1 \prec \sigma 2 \vee \sigma 1 = \sigma 2 \wedge$ *nfinite* $\sigma 1\ )$

definition *finitary*
where *finitary* $S = (\forall\ \sigma.\ \sigma \in S \longrightarrow$ *nfinite* $\sigma)$

definition *infinitary*
where *infinitary* $S = (\forall\ \sigma.\ \sigma \in S \longrightarrow \neg$ *nfinite* $\sigma\ )$

definition *Pref* :: $'a$ *nellist set* $\Rightarrow$ $'a$ *nellist set*
where *Pref* $S = \{\sigma\ .\quad (\exists\ \sigma 1.\ \sigma \preceq \sigma 1\ \wedge \sigma 1 \in S)\ \}$

definition *Fin*
where *Fin* $= \{\sigma\ .$ *nfinite* $\sigma\}$

definition *Inf*
where *Inf* $= \{\sigma\ . \neg$ *nfinite* $\sigma\}$

definition *Empty*
where *Empty* $= \{\sigma\ .$ *nlength* $\sigma = 0\}$

definition *More*
where *More* $= \{\sigma\ .$ *nlength* $\sigma > 0\}$

definition *Ts*
where *Ts* $= \{\sigma\ .$ *nfinite* $\sigma \vee \neg$ *nfinite* $\sigma\}$

definition *Fs*
where *Fs* $= \{\}$

definition *Ap*
where *Ap* $S = \{\ \sigma\ .\ (\forall\ \sigma 1.\ (\sigma 1 \preceq \sigma) \longrightarrow \sigma 1 \in S)\}$

definition *Ep*
where *Ep* $S = \{\sigma\ .\ (\exists\ \sigma 1\ .\ \sigma 1 \preceq \sigma \wedge \sigma 1 \in S)\ \}$

definition *Rp*
where *Rp* $S = \{\sigma\ .\ (\forall\ \sigma 1\ .\ \sigma 1 \preceq \sigma \longrightarrow (\exists\ \sigma 2.\ \sigma 1 \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \wedge \sigma 2 \in S))\ \}$

definition *Pp*
where *Pp* $S = \{\sigma\ .\ (\exists\ \sigma 1\ .\ \sigma 1 \preceq \sigma \wedge (\forall\ \sigma 2.\ \sigma 1 \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \longrightarrow \sigma 2 \in S))\ \}$

definition *Ap-f*
where *Ap-f* $S =$ *Ap* $S \cap$ *Fin*

definition *Ap-i*
where *Ap-i* $S =$ *Ap* $S \cap$ *Inf*

definition *Ep-f*
where *Ep-f S* = *Ep S* ∩ *Fin*

definition *Ep-i*
where *Ep-i S* = *Ep S* ∩ *Inf*

definition *Rp-f*
where *Rp-f S* = *Rp S* ∩ *Fin*

definition *Rp-i*
where *Rp-i S* = *Rp S* ∩ *Inf*

definition *Pp-f*
where *Pp-f S* = *Pp S* ∩ *Fin*

definition *Pp-i*
where *Pp-i S* = *Pp S* ∩ *Inf*

definition *safety-prop*
where *safety-prop S* = (∃ *S1*. *finitary S1* ∧ *infinitary S* ∧ *S* = (*Ap-i S1*))

definition *guarantee-prop*
where *guarantee-prop S* = (∃ *S1*. *finitary S1* ∧ *infinitary S* ∧ *S* = (*Ep-i S1*))

definition *response-prop*
where *response-prop S* = (∃ *S1*. *finitary S1* ∧ *infinitary S* ∧ *S* = (*Rp-i S1*))

definition *persistence-prop*
where *persistence-prop S* = (∃ *S1*. *finitary S1* ∧ *infinitary S* ∧ *S* = (*Pp-i S1*))

definition *finitary-list*
where *finitary-list LS* = *foldr*(λ *S1 LS2*. *finitary S1* ∧ *LS2*) *LS True*

definition *infinitary-list*
where *infinitary-list LS* = *foldr*(λ *S1 LS2*. *infinitary S1* ∧ *LS2*) *LS True*

definition *inter-list*
where *inter-list LI* = *foldr*(λ *I LI1*. *I* ∩ *LI1*) *LI UNIV*

definition *union-list*
where *union-list LI* = *foldr*(λ *I LI1*. *I* ∪ *LI1*) *LI* {}

definition *obligation-prop*
where *obligation-prop S k* =
(∃ *LS1 LS2*. *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *0* < *length LS1* ∧ *length LS1* = *k* ∧ *infinitary S* ∧
S = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*))

definition *reactivity-prop*
where *reactivity-prop S k* =

$(\exists\ LS1\ LS2.\ finitary\text{-}list\ LS1 \wedge finitary\text{-}list\ LS2 \wedge$
$length\ LS1 = length\ LS2 \wedge 0 < length\ LS1 \wedge length\ LS1 = k \wedge infinitary\ S \wedge$
$S = inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ (Rp\text{-}i\ S1) \cup (Pp\text{-}i\ S2))\ LS1\ LS2))$

definition *cat* (**infix** $\cdot$ *65*)
where $cat\ S1\ S2 = \{\ \sigma\ .\ (\exists\ \sigma 1\ \sigma 2.\ \sigma = nappend\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in S1 \wedge \sigma 2 \in S2)\}$

definition *wcat* (**infix** $\odot$ *65*)
where $wcat\ S1\ S2 = (\ S1) \cup (S1 \cdot S2)$

definition *Since* (**infix** $\mathcal{S}$ *65*)
where $Since\ S1\ S2 = \{\ \sigma\ .\ nfinite\ \sigma \wedge (\exists \sigma 1\ .\ (\sigma 1 \preceq \sigma) \wedge \sigma 1 \in S2 \wedge$
$(\forall\ \sigma 2.\ \sigma 1 \prec \sigma 2 \wedge \sigma 2 \preceq \sigma \longrightarrow \sigma 2 \in S1))\ \}$

definition *and-prev-since*
where $and\text{-}prev\text{-}since\ P1\ P2 =$
$(\lambda\ k1{::}nat.\ P1\ k1 \wedge$
$(\exists\ k2{::}nat.\ k2 < k1 \wedge P2\ k2 \wedge$
$(\forall k22.\ \ k2 < k22 \wedge k22 < k1\ \longrightarrow \neg\ (P1\ k22))))$

definition *pfirst*
where $pfirst\ P = (LEAST\ (n{::}nat).\ \ P\ n)$

definition *pleast*
where $pleast\ kl\ P = (LEAST\ (n{::}nat).\ kl < n \wedge P\ n)$

definition *pleast-niterates*
where $pleast\text{-}niterates\ P = niterates\ (\lambda\ s.\ pleast\ s\ P)\ (pfirst\ P)$

definition *infinite-list-of*
where $infinite\text{-}list\text{-}of\ P\ nl =$
$(\neg nfinite\ (nl :: nat\ nellist) \wedge$
$(\forall\ i.\ P\ (nnth\ nl\ i)) \wedge$
$(\forall\ i\ .\ nnth\ nl\ i < nnth\ nl\ (Suc\ i)) \wedge$
$(\forall\ l.\ l < nnth\ nl\ 0 \longrightarrow \neg\ P\ l) \wedge$
$(\forall\ i\ l.\ \ nnth\ nl\ i < l \wedge l <\ \ nnth\ nl\ (Suc\ i) \longrightarrow \neg\ P\ l))$

definition *w-infinite-list-of*
where $w\text{-}infinite\text{-}list\text{-}of\ P\ nl =$
$(\neg nfinite\ (nl :: nat\ nellist) \wedge$
$(\forall\ i.\ P\ (nnth\ nl\ i)) \wedge$
$(\forall\ i\ .\ nnth\ nl\ i < nnth\ nl\ (Suc\ i))\ )$

definition *add-left*

where *add-left a LS1* = (*map* (λ *S*. *S* ∪ *a*) *LS1*)@*LS1*

definition *add-left-p*
where *add-left-p a LS1* = (*map* (λ *S*. *S* ∪ (*Fin* − (((*S* 𝒮 (*Fin* − *a*)) · *Empty*)))) *LS1*)@*LS1*

definition *add-right*
where *add-right b LS2* = (*LS2*@(*map* (λ*S*. *Ap-f S* ∪ *Ap-f b*) *LS2*))

definition *add-right-r*
where *add-right-r b LS2* = (*LS2*@(*map* (λ*S*. *S* ∪ *b*) *LS2*))

primrec *add-left-list*
where *add-left-list* [] *LS1* = *LS1* |
add-left-list (*a* # *LS3*) *LS1* = (*add-left-list LS3* (*add-left a LS1*))

primrec *add-left-p-list*
where *add-left-p-list* [] *LS1* = *LS1* |
add-left-p-list (*a* # *LS3*) *LS1* = (*add-left-p-list LS3* (*add-left-p a LS1*))

primrec *add-right-list*
where *add-right-list* [] *LS2* = *LS2* |
add-right-list (*b* # *LS4*) *LS2* = (*add-right-list LS4* (*add-right b LS2*))

primrec *add-right-r-list*
where *add-right-r-list* [] *LS2* = *LS2* |
add-right-r-list (*b* # *LS4*) *LS2* = (*add-right-r-list LS4* (*add-right-r b LS2*))

7.8 Lemmas

7.8.1 nprefix and nprefix_eq

lemma *nprefix-eq-exists-ntaken*:
σ1 ≼ *σ2* ⟷ *nfinite σ1* ∧ (∃ *k*. *k*≤ *nlength σ2* ∧ *σ1* = *ntaken k σ2*)
apply (*simp add*: *nprefix-eq-def nprefix-def*)
by (*metis nfinite-nlength-enat nfuse-def1 nfuse-ntaken-ndropn nle-le ntaken-all ntaken-nappend1*)

lemma *nprefix-exists-ntaken*:
σ1 ≺ *σ2* ⟷ *nfinite σ1* ∧ (∃ *k*. *k*< *nlength σ2* ∧ *σ1* = *ntaken k σ2*)
apply (*auto simp add*: *nprefix-eq-def nprefix-def*)
apply (*metis enat-le-plus-same*(*1*) *iless-Suc-eq ndropn-eq-NNil ndropn-nlast nfinite-conv-nlength-enat ntaken-all ntaken-nappend1 plus-1-eSuc*(*2*) *the-enat.simps*)
by (*metis Suc-ile-eq nappend-ntaken-ndropn*)

lemma *nprefix-refl*:
nfinite σ ⟹ *σ* ≼ *σ*
by (*simp add*: *nprefix-eq-def*)

lemma *nprefix-trans*:
assumes *nfinite σ1*
nfinite σ2

$\sigma 1 \preceq \sigma 2$
$\sigma 2 \preceq \sigma 3$
shows $\sigma 1 \preceq \sigma 3$
using *assms*
apply (*simp add*: *nprefix-eq-def nprefix-def*)
using *nappend-assoc* **by** *blast*

lemma *nprefix-ntaken*:
assumes $k \leq$ *nlength* σ
shows (*ntaken k* σ) $\preceq \sigma$
using *assms*
using *nfinite-ntaken nprefix-eq-exists-ntaken* **by** *blast*

lemma *nprefix-ntaken-ntaken*:
assumes $k+m \leq$ *nlength* σ
shows (*ntaken k* σ) $\preceq$ *ntaken* $(m+k)$ σ
using *assms*
apply (*simp add*: *nprefix-eq-def nprefix-def*)
by (*metis add.commute nle-le nprefix-def nprefix-eq-def ntaken-all*
pref-ntaken-3 nprefix-ntaken)

lemma *nprefix-ntaken-ntaken-alt*:
assumes $m \leq$ *nlength* σ
$k \leq m$
shows (*ntaken k* σ) $\preceq$ *ntaken m* σ
using *assms*
by (*metis min.orderE nfinite-ntaken nle-le nprefix-eq-def*
ntaken-all ntaken-ntaken nprefix-eq-exists-ntaken)

lemma *nprefix-ntaken-infinite*:
assumes $\neg$ *nfinite* σ
shows (*ntaken k* σ) $\preceq \sigma$
by (*metis assms linorder-le-cases nfinite-ntaken ntaken-all*
nprefix-ntaken)

lemma *nprefix-nappend*:
assumes *nfinite* σ
shows $\sigma \preceq$ *nappend* σ $\sigma 1$
unfolding *nprefix-eq-def nprefix-def*
using *assms* **by** *auto*

7.8.2 Pref, finitary, finitary_list, infinitary_list, inter_list and union_list

lemma *nfinite-in-Pref*:
$\sigma \in$ *Pref S* $\Longrightarrow$ *nfinite* σ
unfolding *Pref-def nprefix-eq-def nprefix-def*
by *auto*

lemma *Pref-Inf-eq-Fin*:

Pref Inf = Fin
unfolding *Pref-def Inf-def Fin-def*
by (*metis CollectI nfinite-nappend nfinite-niterates*
nprefix-eq-exists-ntaken nprefix-nappend)

lemma *Pref-Fin-eq-Fin*:
Pref Fin = Fin
unfolding *Pref-def Fin-def*
using *nprefix-eq-exists-ntaken nprefix-refl* **by** *blast*

lemma *Pref-mono*:
assumes $A \subseteq B$
shows *Pref A* $\subseteq$ *Pref B*
using *assms* **unfolding** *Pref-def* **by** *auto*

lemma *Pref-union*:
Pref A $\cup$ *Pref B* = *Pref* $(A \cup B)$
unfolding *Pref-def* **by** *auto*

lemma *Pref-Pref*:
Pref (*Pref A*) = *Pref A*
unfolding *Pref-def* **apply** *auto*
apply (*metis nprefix-eq-exists-ntaken nprefix-trans*)
using *nprefix-eq-exists-ntaken nprefix-refl* **by** *blast*

lemma *Pref-inter*:
Pref A $\cap$ *Pref B* = *Pref* (*Pref A* $\cap$ *Pref B*)
unfolding *Pref-def* **apply** *auto*
using *nprefix-eq-exists-ntaken nprefix-refl* **apply** *blast*
apply (*meson nprefix-eq-exists-ntaken nprefix-trans*)
by (*meson nprefix-eq-exists-ntaken nprefix-trans*)

lemma *Pref-defs*:
assumes *finitary A*
shows $x \in$ *Pref* $(A \cdot B) \longleftrightarrow$
$(\exists \sigma 1.\ (nfinite\ x \wedge (\exists \sigma 3.\ \sigma 1 = nappend\ x\ \sigma 3) \vee x = \sigma 1 \wedge nfinite\ x) \wedge$
$(\exists \sigma 1a\ \sigma 2.\ \sigma 1 = nappend\ \sigma 1a\ \sigma 2 \wedge nfinite\ \sigma 1a \wedge \sigma 1a \in A \wedge \sigma 2 \in B))$
using *assms* **unfolding** *Pref-def cat-def nprefix-eq-def nprefix-def*
by *simp*

lemma *Pref-Empty*:
Pref Empty = Empty
unfolding *Pref-def cat-def Empty-def nprefix-eq-def nprefix-def*
apply *auto*
by (*metis nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *subseteq-Fin-eq-finitary*:
$S \subseteq Fin \longleftrightarrow finitary\ S$
unfolding *Fin-def finitary-def* **by** *auto*

lemma *subseteq-Inf-eq-infinitary*:
$S \subseteq Inf \longleftrightarrow infinitary\ S$
unfolding *Inf-def infinitary-def* **by** *auto*

lemma *finitary-Pref*:
finitary (*Pref S*)
using *finitary-def nfinite-in-Pref* **by** *auto*

lemma *finitary-Fin-complement*:
finitary (*Fin* − *S*)
by (*simp add*: *Fin-def finitary-def*)

lemma *finitary-Fin*:
finitary Fin
unfolding *finitary-def Fin-def* **by** *simp*

lemma *finitary-inter*:
assumes *finitary S*
shows $S \cap Fin = S$
using *assms* **unfolding** *finitary-def Fin-def*
by *auto*

lemma *finitary-Fs*:
finitary Fs
by (*simp add*: *Fs-def finitary-def*)

lemma *Ts-Eqv-Inf-Fin*:
$Ts = Inf \cup Fin$
unfolding *Ts-def Inf-def Fin-def* **by** *auto*

lemma *Fin-minus-Fin*:
$Fin - Fin = Fs$
unfolding *Fin-def Fs-def* **by** *auto*

lemma *finitary-list-defs*:
$finitary\text{-}list\ LS \longleftrightarrow (\forall k.\ k < length\ LS \longrightarrow finitary\ (LS!k))$
proof (*induct LS*)
case *Nil*
then show *?case* **unfolding** *finitary-list-def* **by** *simp*
next
case (*Cons a LS*)
then show *?case*
proof −
have *1*: $finitary\text{-}list\ (a\ \#\ LS) \longleftrightarrow finitary\ a \wedge finitary\text{-}list\ (LS)$

```
    unfolding finitary-list-def by simp
  have 3: (∀ k<length (a # LS). finitary ((a #LS) ! k)) ⟷
        finitary a ∧ (∀ k<length ( LS). finitary (LS ! k))
    by (metis list.pred-inject(2) list-all-length)
  show ?thesis
  using 1 3 local.Cons by presburger
 qed
qed

lemma finitary-list-append:
 shows finitary-list (LS1@LS2) ⟷ finitary-list LS1 ∧ finitary-list LS2
proof (induction LS1 arbitrary: LS2)
case Nil
then show ?case unfolding finitary-list-def by simp
next
case (Cons a LS1)
then show ?case unfolding finitary-list-def by simp
qed


lemma infinitary-list-defs:
 infinitary-list LS ⟷ (∀ k. k < length LS ⟶ infinitary (LS!k))
proof (induct LS)
case Nil
then show ?case unfolding infinitary-list-def by simp
next
case (Cons a LS)
then show ?case
 proof −
  have 1: infinitary-list (a # LS) ⟷ infinitary a ∧ infinitary-list (LS)
    unfolding infinitary-list-def by simp
  have 3: (∀ k<length (a # LS). infinitary ((a #LS) ! k)) ⟷
        infinitary a ∧ (∀ k<length ( LS). infinitary (LS ! k))
    by (metis list.pred-inject(2) list-all-length)
  show ?thesis using 1 3 local.Cons by presburger
 qed
qed

lemma infinitary-list-append:
 shows infinitary-list (LS1@LS2) ⟷ infinitary-list LS1 ∧ infinitary-list LS2
proof (induction LS1 arbitrary: LS2)
case Nil
then show ?case unfolding infinitary-list-def by simp
next
case (Cons a LS1)
then show ?case unfolding infinitary-list-def by simp
qed

lemma inter-list-defs:
  inter-list LI = (⋂ x ∈ {s . s < length LI}. (LI ! x))
```

proof (*induction LI*)
case *Nil*
then show *?case* **unfolding** *inter-list-def* **by** *simp*
next
case (*Cons a LI*)
then show *?case*
proof −
have *1*: *inter-list* (*a* # *LI*) = *a* ∩ *inter-list LI*
unfolding *inter-list-def* **by** *auto*
have *11*: ⋀*x xa*. ∀ *xa*<*Suc* (*length LI*). *x* ∈ (*a* # *LI*) ! *xa* ⟹ *xa* < *length LI* ⟹ *x* ∈ *LI* ! *xa*
by *auto*
have *12*: ⋀*x xa*. *x* ∈ *a* ⟹ ∀ *xa*<*length LI*. *x* ∈ *LI* ! *xa* ⟹ *xa* < *Suc* (*length LI*) ⟹ *x* ∈ (*a* # *LI*) ! *xa*
using *less-Suc-eq-0-disj* **by** *fastforce*
have *2*: ⋂ ((!) (*a* # *LI*) ‘ {*s*. *s* < *length* (*a* # *LI*)}) =
a ∩ ⋂ ((!) (*LI*) ‘ {*s*. *s* < *length* (*LI*)})
using *11 12* **by** *auto*
show *?thesis*
using *1 2 local.Cons* **by** *presburger*
qed
qed

lemma *inter-list-append*:
shows *inter-list* (*LI1*@*LI2*) = *inter-list LI1* ∩ *inter-list LI2*
proof (*induct LI1 arbitrary*: *LI2*)
case *Nil*
then show *?case* **unfolding** *inter-list-def* **by** *simp*
next
case (*Cons a LI1*)
then show *?case* **unfolding** *inter-list-def* **by** *auto*
qed

lemma *union-list-defs*:
union-list LI = (⋃ *x* ∈ {*s* . *s* < *length LI*}. (*LI* ! *x*))
proof (*induction LI*)
case *Nil*
then show *?case* **unfolding** *union-list-def* **by** *simp*
next
case (*Cons a LI*)
then show *?case*
proof −
have *1*: *union-list* (*a* # *LI*) = *a* ∪ *union-list LI*
unfolding *union-list-def* **by** *auto*
have *11*: ⋀*x xa*. *x* ∈ (*a* # *LI*) ! *xa* ⟹ ∀ *xa*<*length LI*. *x* ∉ *LI* ! *xa* ⟹ *xa* < *Suc* (*length LI*) ⟹ *x* ∈ *a*
by (*metis Suc-diff-1 bot-nat-0.not-eq-extremum not-less-eq nth-Cons-0 nth-Cons-pos*)
have *2*: ⋃ ((!) (*a* # *LI*) ‘ {*s*. *s* < *length* (*a* # *LI*)}) =
a ∪ ⋃ ((!) (*LI*) ‘ {*s*. *s* < *length* (*LI*)})
using *11* **by** *auto*
show *?thesis*
using *1 2 local.Cons* **by** *presburger*

qed
qed

lemma *union-list-append*:
shows *union-list* (*LI1*@*LI2*) = *union-list LI1* ∪ *union-list LI2*
proof (*induct LI1 arbitrary*: *LI2*)
case *Nil*
then show *?case* **unfolding** *union-list-def* **by** *simp*
next
case (*Cons a LI1*)
then show *?case* **unfolding** *union-list-def* **by** *auto*
qed

lemma *complement-inter-list*:
assumes *0*<*length LS*
shows *Inf* − *inter-list LS* = *union-list* (*map* (λ *S*. *Inf* − *S*) *LS*)
proof −
have *1*: *Inf* − *inter-list LS* = *Inf* − (⋂ *x* ∈ {*s* . *s* < *length LS*}. (*LS* ! *x*))
using *inter-list-defs* **by** *metis*
have *2*: *Inf* − (⋂ *x* ∈ {*s* . *s* < *length LS*}. (*LS* ! *x*)) =
(⋃ *x* ∈ {*s* . *s* < *length LS*}. *Inf* − (*LS* ! *x*))
by *auto*
have *3*: (⋃ *x* ∈ {*s* . *s* < *length LS*}. *Inf* − (*LS* ! *x*)) =
union-list (*map* (λ *S*. *Inf* − *S*) *LS*)
using *union-list-defs*
by (*metis* (*mono-tags*, *lifting*) *Collect-cong SUP-cong length-map mem-Collect-eq nth-map*)
show *?thesis*
using *1 2 3* **by** *presburger*
qed

lemma *complement-union-list*:
assumes *0*<*length LS*
shows *Inf* − *union-list LS* = *inter-list* (*map* (λ *S*. *Inf* − *S*) *LS*)
proof −
have *1*: *Inf* − *union-list LS* = *Inf* − (⋃ *x* ∈ {*s* . *s* < *length LS*}. (*LS* ! *x*))
using *union-list-defs* **by** *metis*
have *2*: *Inf* − (⋃ *x* ∈ {*s* . *s* < *length LS*}. (*LS* ! *x*)) =
(⋂ *x* ∈ {*s* . *s* < *length LS*}. *Inf* − (*LS* ! *x*))
apply *auto*
using *assms* **by** *auto*
have *3*: (⋂ *x* ∈ {*s* . *s* < *length LS*}. *Inf* − (*LS* ! *x*)) =
inter-list (*map* (λ *S*. *Inf* − *S*) *LS*)
using *inter-list-defs*
by (*metis* (*mono-tags*, *lifting*) *Collect-cong Sup.SUP-cong length-map mem-Collect-eq nth-map*)
show *?thesis*
using *1 2 3* **by** *presburger*
qed

7.8.3 cat and wcat

lemma *prev-defs*:
assumes *finitary S*
shows $\sigma \in S \cdot Empty \longleftrightarrow nfinite\ \sigma \wedge (\exists\ \sigma 1\ \sigma 2.\ nlength\ \sigma 2 = 0 \wedge \sigma 1 \prec \sigma \wedge \sigma = nappend\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in S)$
using *assms* **unfolding** *cat-def Empty-def* **apply** *simp*
by (*metis enat-0-iff*(2) *nfinite-nappend nlength-eq-enat-nfiniteD nprefix-def*)

lemma *prev-defs-alt*:
assumes *finitary S*
shows $\sigma \in S \cdot Empty \longleftrightarrow nfinite\ \sigma \wedge nlength\ \sigma > 0 \wedge (ntaken\ (the\text{-}enat\ (nlength\ \sigma) - 1)\ \sigma) \in S$
using *assms* **unfolding** *finitary-def cat-def Empty-def*
apply *auto*
apply (*simp add*: *nfinite-nappend nlength-eq-enat-nfiniteD zero-enat-def*)
apply (*metis One-nat-def diff-Suc-1 eSuc-enat nfinite-nlength-enat ntaken-all ntaken-nappend1 order-le-less plus-1-eSuc*(2) *the-enat.simps*)
using *One-nat-def nellist-split-2-last nfinite-conv-nlength-enat nlength-NNil not-gr-zero the-enat.simps zero-enat-def*
by *metis*

lemma *wprev-defs*:
assumes *finitary S*
shows $\sigma \in S \odot Empty \longleftrightarrow nfinite\ \sigma \wedge ((\sigma \in S) \vee (\exists\ \sigma 1\ \sigma 2.\ nlength\ \sigma 2 = 0 \wedge \sigma 1 \prec \sigma \wedge \sigma = nappend\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in S))$
using *assms* **unfolding** *finitary-def wcat-def cat-def Empty-def Fin-def*
apply *auto*
apply (*simp add*: *nfinite-nappend nlength-eq-enat-nfiniteD zero-enat-def*)
using *nprefix-def* **by** *blast*

lemma *wprev-defs-alt*:
assumes *finitary S*
shows $\sigma \in S \odot Empty \longleftrightarrow nfinite\ \sigma \wedge ((\sigma \in S) \vee nlength\ \sigma > 0 \wedge (ntaken\ (the\text{-}enat\ (nlength\ \sigma) - 1)\ \sigma) \in S)$
using *assms*
unfolding *wcat-def* **using** *prev-defs-alt*[*of S* σ]
by (*metis Un-iff wcat-def wprev-defs*)

7.8.4 since

lemma *in-since-defs*:
$\sigma \in S1\ \mathcal{S}\ S2 \longleftrightarrow$
$nfinite\ \sigma \wedge (\exists \sigma 1\ .\ (\sigma 1 \preceq \sigma) \wedge \sigma 1 \in S2 \wedge (\forall\ \sigma 2.\ \sigma 1 \prec \sigma 2 \wedge \sigma 2 \preceq \sigma \longrightarrow \sigma 2 \in S1))$
unfolding *Since-def* **by** *blast*

lemma *in-since-defs-alt-help-1*:
assumes *nfinite* σ
$enat\ k \leq nlength\ \sigma$
$ntaken\ k\ \sigma \in S2$
$\forall\, l.\ k < l \wedge enat\ l \leq nlength\ \sigma \longrightarrow ntaken\ l\ \sigma \in S1$
shows $\exists \sigma 1.\ \sigma 1 \preceq \sigma \wedge \sigma 1 \in S2 \wedge (\forall \sigma 2.\ \sigma 1 \prec \sigma 2 \wedge \sigma 2 \preceq \sigma \longrightarrow \sigma 2 \in S1)$

proof −
have *1*: *ntaken k σ ⪯ σ*
by (*simp add*: *assms*(*2*) *nprefix-ntaken*)
have *14*: ⋀*l*. *nfinite σ ⟹*
enat k ≤ nlength σ ⟹
ntaken k σ ∈ S2 ⟹ ∀ *l*. *k < l* ∧ *enat l ≤ nlength σ ⟶ ntaken l σ ∈ S1 ⟹ ntaken k σ ≺ ntaken l σ ⟹*
ntaken l σ ⪯ σ ⟹ ntaken l σ ∈ S1
by (*metis enat-ord-simps*(*2*) *min.absorb3 min-def nprefix-eq-exists-ntaken nprefix-exists-ntaken ntaken-nlength*)
have *15*: (∀ *l*. *ntaken k σ ≺ ntaken l σ* ∧ *ntaken l σ ⪯ σ ⟶ ntaken l σ ∈ S1*)
using *assms 14* **by** *auto*
have *2*: (∀ *σ2*. *ntaken k σ ≺ σ2* ∧ *σ2 ⪯ σ ⟶ σ2 ∈ S1*)
by (*metis 15 nprefix-eq-exists-ntaken*)
show *?thesis*
using *1 2 assms*(*3*) **by** *auto*
qed

lemma *in-since-defs-alt-help-2*:
assumes *nfinite σ*
σ1 ⪯ σ
σ1 ∈ S2
∀ *σ2*. *σ1 ≺ σ2* ∧ *σ2 ⪯ σ ⟶ σ2 ∈ S1*
shows ∃ *k*. *enat k ≤ nlength σ* ∧ *ntaken k σ ∈ S2* ∧ (∀ *l*. *k < l* ∧ *enat l ≤ nlength σ ⟶ ntaken l σ ∈ S1*)
using *assms nprefix-eq-exists-ntaken*[*of - σ*] *nprefix-exists-ntaken*[*of σ1 -*]
by (*metis enat-ord-simps*(*2*) *min.orderE not-less-iff-gr-or-eq nprefix-def nprefix-ntaken ntaken-nappend2 ntaken-nlength order-le-less*)

lemma *in-since-defs-alt*:
σ ∈ S1 𝒮 S2 ⟷
nfinite σ ∧ (∃ *k*. *k ≤ nlength σ* ∧ (*ntaken k σ*) ∈ *S2* ∧ (∀ *l*. *k < l* ∧ *l ≤ nlength σ ⟶* (*ntaken l σ*) ∈ *S1*))
unfolding *Since-def*
apply *auto*
using *in-since-defs-alt-help-2*[*of σ - S2 S1*]
apply *blast*
using *in-since-defs-alt-help-1*[*of σ - S2 S1*]
by *blast*

lemma *prev-since-defs*:
assumes *finitary S1*
finitary S2
shows *σ ∈* (*S1 𝒮 S2*) · *Empty ⟷*
nfinite σ ∧ (∃ *σ1* . (*σ1 ≺ σ*) ∧ *σ1 ∈ S2* ∧ (∀ *σ2*. *σ1 ≺ σ2* ∧ *σ2 ≺ σ ⟶ σ2 ∈ S1*))
proof −
have *1*: *finitary* (*S1 𝒮 S2*)
by (*simp add*: *finitary-def in-since-defs*)
have *2*: *σ ∈* (*S1 𝒮 S2*) · *Empty ⟷*

```
        (nfinite σ ∧ (∃σ1 σ2. nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧ σ1 ∈ S1 𝒮 S2))
 using 1  prev-defs[of (S1 𝒮 S2) σ] by blast
have 3: ⋀ σ1. σ1 ∈ S1 𝒮 S2 ⟷
          (nfinite σ1 ∧ (∃σ3. σ3 ⪯ σ1 ∧ σ3 ∈ S2 ∧ (∀σ4. σ3 ≺ σ4 ∧ σ4 ⪯ σ1 ⟶ σ4 ∈ S1)))
   by (simp add: in-since-defs)
have 4: (nfinite σ ∧ (∃σ1 σ2. nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧ σ1 ∈ S1 𝒮 S2)) ⟷
        (nfinite σ ∧ (∃σ1 σ2. nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧
         (nfinite σ1 ∧ (∃σ3. σ3 ⪯ σ1 ∧ σ3 ∈ S2 ∧ (∀σ4. σ3 ≺ σ4 ∧ σ4 ⪯ σ1 ⟶ σ4 ∈ S1)))))
using  3 by auto
have 5: (nfinite σ ∧ (∃σ1 σ2. nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧
         (nfinite σ1 ∧ (∃σ3. σ3 ⪯ σ1 ∧ σ3 ∈ S2 ∧ (∀σ4. σ3 ≺ σ4 ∧ σ4 ⪯ σ1 ⟶ σ4 ∈ S1))))) ⟹
       nfinite σ ∧ (∃σ1 . (σ1 ≺ σ) ∧ σ1 ∈ S2 ∧ (∀ σ2. σ1 ≺ σ2 ∧ σ2 ≺ σ ⟶ σ2 ∈ S1))
  proof −
   assume a: (nfinite σ ∧ (∃σ1 σ2. nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧
         (nfinite σ1 ∧ (∃σ3. σ3 ⪯ σ1 ∧ σ3 ∈ S2 ∧ (∀σ4. σ3 ≺ σ4 ∧ σ4 ⪯ σ1 ⟶ σ4 ∈ S1)))))
   show nfinite σ ∧ (∃σ1 . (σ1 ≺ σ) ∧ σ1 ∈ S2 ∧ (∀ σ2. σ1 ≺ σ2 ∧ σ2 ≺ σ ⟶ σ2 ∈ S1))
   proof −
    obtain σ1 σ2 σ3 where 6: nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧
         (nfinite σ1 ∧ ( σ3 ⪯ σ1 ∧ σ3 ∈ S2 ∧ (∀σ4. σ3 ≺ σ4 ∧ σ4 ⪯ σ1 ⟶ σ4 ∈ S1)))
     using a by blast
     have 7: nfinite σ
       using a by blast
     have 8: σ3 ≺ σ
      by (metis 6 nappend-assoc nprefix-def nprefix-eq-def)
     have 9: σ3 ∈ S2
       using 6 by blast
     have 10: (∀ σ2. σ3 ≺ σ2 ∧ σ2 ≺ σ ⟶ σ2 ∈ S1)  using  6 unfolding nprefix-eq-exists-ntaken
      apply simp
      by (metis add.right-neutral iless-Suc-eq nlength-nappend nprefix-exists-ntaken ntaken-nappend1
        plus-1-eSuc(2))
     show ?thesis
     using 10 7 8 9 by auto
    qed
   qed
  have 11:  nfinite σ ∧ (∃σ1 . (σ1 ≺ σ) ∧ σ1 ∈ S2 ∧ (∀ σ2. σ1 ≺ σ2 ∧ σ2 ≺ σ ⟶ σ2 ∈ S1)) ⟹
            (nfinite σ ∧ (∃σ1 σ2. nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧
           (nfinite σ1 ∧ (∃σ3. σ3 ⪯ σ1 ∧ σ3 ∈ S2 ∧ (∀σ4. σ3 ≺ σ4 ∧ σ4 ⪯ σ1 ⟶ σ4 ∈ S1)))))
  proof −
   assume b:  nfinite σ ∧ (∃σ1 . (σ1 ≺ σ) ∧ σ1 ∈ S2 ∧ (∀ σ2. σ1 ≺ σ2 ∧ σ2 ≺ σ ⟶ σ2 ∈ S1))
   show (nfinite σ ∧ (∃σ1 σ2. nlength σ2 = 0 ∧ σ1 ≺ σ ∧ σ = nappend σ1 σ2 ∧
         (nfinite σ1 ∧ (∃σ3. σ3 ⪯ σ1 ∧ σ3 ∈ S2 ∧ (∀σ4. σ3 ≺ σ4 ∧ σ4 ⪯ σ1 ⟶ σ4 ∈ S1)))))
    using b
   proof −
    obtain σ1 where 12:  nfinite σ ∧ ( (σ1 ≺ σ) ∧ σ1 ∈ S2 ∧ (∀ σ2. σ1 ≺ σ2 ∧ σ2 ≺ σ ⟶ σ2 ∈ S1))

     using b by blast
    have 13: (∃ σ11 σ21. nlength σ21 = 0 ∧ σ11 ≺ σ ∧ σ = nappend σ11 σ21 )
       by (metis 12 is-NNil-nappend nappend-nbutlast-nlast-id nbutlast-nfinite nlength-NNil nprefix-def)
    obtain σ11 σ21 where 14: nlength σ21 = 0 ∧ σ11 ≺ σ ∧ σ = nappend σ11 σ21
     using 13 by auto
```

have *15*: *nfinite σ11*
using *14 nprefix-def* **by** *auto*
have *16*: *σ1* $\preceq$ *σ11* **using** *12 14* **unfolding** *nprefix-eq-exists-ntaken nprefix-def* **apply** *simp*
by (*metis 12 add.right-neutral iless-Suc-eq nlength-nappend nprefix-exists-ntaken ntaken-nappend1 plus-1-eSuc*(*2*))

have *17*: ($\forall$ *σ4*. *σ1* $\prec$ *σ4* $\wedge$ *σ4* $\preceq$ *σ11* $\longrightarrow$ *σ4* $\in$ *S1*)
by (*metis 12 14 nappend-assoc nprefix-def nprefix-eq-def*)
show *?thesis*
using *12 14 16 17 nprefix-def* **by** *blast*
qed
qed
show *?thesis*
using *11 2 4 5* **by** *argo*
qed

lemma *in-prev-since*:
assumes *finitary S1*
finitary S2
¬*nfinite σ*
shows (*ntaken k3 σ*) $\in$ ((*Fin* − *S1*) $\mathcal{S}$ *S2*) · *Empty* $\longleftrightarrow$
($\exists$ *k2*. *k2* < *k3* $\wedge$ *ntaken k2 σ* $\in$ *S2* $\wedge$
($\forall$ *k22*. *k2* < *k22* $\wedge$ *k22* < *k3* $\longrightarrow$ *ntaken k22 σ* $\notin$ *S1*))
proof −
have *1*: *finitary* ((*Fin* − *S1*) $\mathcal{S}$ *S2*)
using *assms finitary-def in-since-defs* **by** *blast*
have *2*: (*ntaken k3 σ* $\in$ ((*Fin* − *S1*) $\mathcal{S}$ *S2*) · *Empty*) =
(*nfinite* (*ntaken k3 σ*) $\wedge$ *0* < *nlength* (*ntaken k3 σ*) $\wedge$
ntaken (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) (*ntaken k3 σ*) $\in$ (*Fin* − *S1*) $\mathcal{S}$ *S2*)
using *assms prev-defs-alt*[*of* ((*Fin* − *S1*) $\mathcal{S}$ *S2*) (*ntaken k3 σ*)] *1* **by** *blast*
have *3*: (*nfinite* (*ntaken k3 σ*) $\wedge$ *0* < *nlength* (*ntaken k3 σ*) $\wedge$
ntaken (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) (*ntaken k3 σ*) $\in$ (*Fin* − *S1*) $\mathcal{S}$ *S2*) $\longleftrightarrow$
(*nfinite* (*ntaken k3 σ*) $\wedge$ *0* < *nlength* (*ntaken k3 σ*) $\wedge$
($\exists$ *k*. *enat k* $\leq$ *nlength* (*ntaken* (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) (*ntaken k3 σ*)) $\wedge$
ntaken k (*ntaken* (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) (*ntaken k3 σ*)) $\in$ *S2* $\wedge$
($\forall$ *l*. *k* < *l* $\wedge$ *enat l* $\leq$ *nlength* (*ntaken* (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) (*ntaken k3 σ*)) $\longrightarrow$
ntaken l (*ntaken* (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) (*ntaken k3 σ*)) $\in$ *Fin* − *S1*)))
using *in-since-defs-alt*[*of ntaken* (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) (*ntaken k3 σ*)
(*Fin* − *S1*) *S2*] **using** *nfinite-ntaken* **by** *blast*
have *31*: *k3* < *nlength σ*
by (*meson assms*(*3*) *enat-iless linorder-less-linear nlength-eq-enat-nfiniteD*)
have *32*: (*ntaken k3 σ* $\in$ ((*Fin* − *S1*) $\mathcal{S}$ *S2*) · *Empty*) $\Longrightarrow$
0<*k3*
using *2 3 31 assms* **unfolding** *finitary-def Fin-def* **apply** *auto*
by (*metis not-gr-zero zero-enat-def*)
have *33*: (*the-enat* (*nlength* (*ntaken k3 σ*)) − *1*) $\leq$ *k3*
using *31* **by** *auto*

have *33*: (*ntaken* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) (*ntaken k3* σ)) =
ntaken (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) σ
using *ntaken-ntaken*[*of* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) *k3* σ]
using *33* **by** *linarith*
have *34*: *nlength* (*ntaken* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) (*ntaken k3* σ)) =
k3 −*1*
by (*metis 33 assms*(*3*) *min.orderE nfinite-ntaken nle-le ntaken-all ntaken-nlength the-enat.simps*)
have *35*: $\bigwedge$ *k*. *k* ≤ *k3*−*1* ⟹
ntaken k (*ntaken* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) (*ntaken k3* σ)) =
ntaken k σ
by (*simp add*: *31*)
have *36*: (*0* < *k3* ∧ (∃ *k*≤*k3* − *Suc 0*. *ntaken* (*min k* (*k3* − *Suc 0*)) σ ∈ *S2* ∧ (∀ *l*. *k* < *l* ∧ *l* ≤ *k3* − *Suc 0* ⟶ *ntaken l* σ ∉ *S1*))) =
(*0* < *k3* ∧ (∃ *k*≤*k3* − *Suc 0*. *ntaken k* σ ∈ *S2* ∧ (∀ *l*. *k* < *l* ∧ *l* ≤ *k3* − *Suc 0* ⟶ *ntaken l* σ ∉ *S1*)))
by (*metis min-def*)
have *37*: (*0* < *k3* ∧ (∃ *k*≤*k3* − *Suc 0*. *ntaken k* σ ∈ *S2* ∧ (∀ *l*. *k* < *l* ∧ *l* ≤ *k3* − *Suc 0* ⟶ *ntaken l* σ ∉ *S1*))) =
(∃ *k2*<*k3*. *ntaken k2* σ ∈ *S2* ∧ (∀ *k22*. *k2* < *k22* ∧ *k22* < *k3* ⟶ *ntaken k22* σ ∉ *S1*))
apply *auto*
apply (*metis Suc-pred less-le-not-le not-le-imp-less not-less-eq-eq*)
by (*metis Suc-pred bot-nat-0.not-eq-extremum less-le-not-le linorder-not-less not-less-eq*)
have *4*: (*nfinite* (*ntaken k3* σ) ∧ *0* < *nlength* (*ntaken k3* σ) ∧
(∃ *k*. *enat k* ≤ *nlength* (*ntaken* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) (*ntaken k3* σ)) ∧
ntaken k (*ntaken* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) (*ntaken k3* σ)) ∈ *S2* ∧
(∀ *l*. *k* < *l* ∧ *enat l* ≤ *nlength* (*ntaken* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) (*ntaken k3* σ)) ⟶
ntaken l (*ntaken* (*the-enat* (*nlength* (*ntaken k3* σ)) − *1*) (*ntaken k3* σ)) ∈ *Fin* − *S1*))) ⟷
(∃ *k2*. *k2* < *k3* ∧ *ntaken k2* σ ∈ *S2* ∧
(∀ *k22*. *k2* < *k22* ∧ *k22* < *k3* ⟶ *ntaken k22* σ ∉ *S1*))
using *assms 31 32 34* **unfolding** *finitary-def Fin-def*
apply (*simp add*: *zero-enat-def*)
using *36 37* **by** *order*
show *?thesis*
using *2 3 4* **by** *presburger*
qed

7.8.5 infinitely many and infinite_list_of

lemma *exists-least-interval*:
assumes (∃ *k*. (*i*::*nat*) < *k* ∧ *k* ≤ (*j* ::*nat*) ∧ *P k*)
shows ∃ *k*. *i* <*k* ∧ *k*≤ *j* ∧ *P k* ∧ (∀ *l*. *i*<*l* ∧ *l*<*k* ⟶ ¬ *P l*)
using *assms*
proof (*induction j*)
case *0*
then show *?case* **by** *blast*
next
case (*Suc k*)
then show *?case*
by (*metis le-Suc-eq less-Suc-eq-le*)
qed

lemma *exists-least-interval-enat*:
assumes ($\exists$ *k*::*nat*. (*i*::*nat*) < *k* $\wedge$ *enat k* $\leq$ (*j* ::*enat*) $\wedge$ *P k*)
shows $\exists$ *k*::*nat*. *i* <*k* $\wedge$ *enat k*$\leq$ *j* $\wedge$ *P k* $\wedge$ ($\forall$ *l*. *i*<*l* $\wedge$ *l*<*k* $\longrightarrow$ $\neg$ *P l*)
proof (*cases j* = ∞)
case *True*
then show *?thesis*
using *assms exists-least-interval*[*of i - P*]
by *auto*
next
case *False*
then show *?thesis*
using *assms exists-least-interval*[*of i - P*]
by (*meson dual-order.refl dual-order.trans enat-ord-simps*(*1*))
qed

lemma *inf-many-before*:
assumes $\forall$ *kl*. $\exists$ *k1*. *kl* $\leq$ *k1* $\wedge$ (*P1* (*k1*::*nat*))
$\forall$ *kl*. $\exists$ *k2*. *kl* $\leq$ *k2* $\wedge$ (*P2* (*k2*::*nat*))
shows ($\forall$ *kl*. ($\exists$ *k1 k2*. *kl* $\leq$ *k1* $\wedge$ *k2* < *k1* $\wedge$ (*P1* (*k1*::*nat*)) $\wedge$ (*P2* (*k2*::*nat*))))
using *assms*
apply *auto*
by (*metis dual-order.trans gt-ex linorder-le-less-linear order-less-le-trans*)

lemma *inf-many-nellist*:
assumes $\forall$ *kl*. $\exists$ *k*. *kl* $\leq$ *k* $\wedge$ (*P* (*k*::*nat*))
shows $\exists$ (*nl* :: *nat nellist*). $\neg$*nfinite nl* $\wedge$ ($\forall$ *i*. (*P* (*nnth nl i*)))
using *assms*
by (*metis funpow-swap1 nfinite-niterates nnth-niterates*)

lemma *inf-many-least*:
assumes $\exists$ *k*::*nat*. *P* (*kl* + *k*)
shows $\exists$ *k*::*nat*. *P* (*kl* + *k*) $\wedge$ ($\forall$ *l*. *0* $\leq$ *l* $\wedge$ *l* < *k* $\longrightarrow$ $\neg$ *P* (*kl* + *l*))
using *assms*
exists-least-interval-enat[*of kl* ∞ *P*]
apply *auto*
by (*metis add.commute bot-nat-0.not-eq-extremum le-add-diff-inverse less-add-same-cancel2 less-diff-conv less-nat-zero-code order-less-imp-le*)

lemma *inf-many-least-1*:
assumes $\forall$ *kl*. $\exists$ *k*::*nat*. *P* (*kl* + *k*)
shows $\exists$ *k*::*nat*. *P* (*kl* + *k*) $\wedge$ ($\forall$ *l*. *0* $\leq$ *l* $\wedge$ *l* < *k* $\longrightarrow$ $\neg$ *P* (*kl* + *l*))
using *assms*

using *inf-many-least* **by** *auto*

lemma *inf-many-least-2*:
shows $(\forall\, kl.\ \exists\, k{::}nat.\ \ P\ (kl{+}k)) \longleftrightarrow$
$\quad (\forall\, kl.\ \exists\, k{::}nat.\ \ P\ (kl{+}k) \wedge (\forall\ \ l.\ 0{\leq}l \wedge l{<}k \longrightarrow \neg\ P\ (kl{+}l)))$
using *inf-many-least* **by** *auto*

lemma *holds-nnth-nfilter-niterates*:
assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
$\quad l \leq nlength\ (nfilter\ (\lambda s.\ P\ s)\ (niterates\ Suc\ 0))$
shows $P\ (nnth\ (nfilter\ (\lambda s.\ P\ s)\ (niterates\ Suc\ 0))\ l)$
using *assms*
by (*metis enat-ord-simps*(3) *in-nset-conv-nnth nkfilter-nnth-aa nlength-niterates nnth-niterates-Suc*)

lemma *nset-niterates-not-empty*:
assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
shows $\exists\, x{\in}nset\ (niterates\ Suc\ 0).\ P\ x$
using *assms*
by (*metis Suc-funpow add.right-neutral enat-ord-simps*(3) *in-nset-conv-nnth nlength-niterates nnth-niterates*)

lemma *nset-niterates-not-finite*:
assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
shows $\neg\ finite\ \{x\ .\ x \in nset\ (niterates\ Suc\ 0) \wedge P\ x\}$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *infinite-nat-iff-unbounded-le le-add1 mem-Collect-eq nnth-niterates nnth-niterates-Suc nset-niterates*)

lemma *set-holds-not-finite*:
assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
shows $\neg\ finite\ \{x\ .\ P\ x\ \}$
using *assms*
by (*metis infinite-nat-iff-unbounded-le le-iff-add mem-Collect-eq*)

lemma *in-set-nset*:
assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
shows $y \in \{x\ .\ P\ x\ \} \longleftrightarrow y \in nset\ (nfilter\ (\lambda s.\ P\ s)\ (niterates\ Suc\ 0))$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Int-iff Suc-funpow add-cancel-left-right mem-Collect-eq nset-nfilter nset-niterates*)

lemma *set-eq-nset*:
assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
shows $\{x\ .\ P\ x\ \} = nset\ (nfilter\ (\lambda s.\ P\ s)\ (niterates\ Suc\ 0))$
using *assms in-set-nset*[*of P*] **apply** *auto*
apply (*metis nfilter-cong*)
by (*metis nfilter-cong*)

lemma *ndistinct-niterates*:
ndistinct (*niterates Suc 0*)
by (*simp add*: *nidx-expand nidx-imp-ndistinct*)

lemma *holds-pfirst*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows *P* (*pfirst P*)
using *assms*
using *LeastI pfirst-def*
by *auto*

lemma *holds-pleast-next*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
P kl
shows *P* (*pleast kl P*)
using *assms*
unfolding *pleast-def*
by (*metis* (*mono-tags*, *lifting*) *LeastI lessI trans-less-add1*)

lemma *pfirst-less-pleast-pfirst*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows (*pfirst P*) < (*pleast* (*pfirst P*) *P*)
using *assms* **unfolding** *pfirst-def pleast-def*
by (*metis* (*lifting*) *LeastI gt-ex trans-less-add1*)

lemma *pleast-less-pleast-pleast*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows (*pleast kl P*) < *pleast* (*pleast kl P*) *P*
using *assms* **unfolding** *pleast-def*
by (*metis* (*lifting*) *LeastI-ex gt-ex trans-less-add1*)

lemma *pleast-niterates-first*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows *nnth* (*pleast-niterates P*) *0* = *pfirst P*
using *assms* **unfolding** *pleast-niterates-def* **by** *simp*

lemma *pleast-niterates-nnth*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows *nnth* (*pleast-niterates P*) *i* = $((\lambda\ s.\ pleast\ s\ P)\widehat{\ }\widehat{\ }i)$ (*pfirst P*)
using *assms*
unfolding *pleast-niterates-def*
by *simp*

lemma *pleast-niterates-Suc-pleast*:

assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows $(((\lambda\ s.\ pleast\ s\ P)\widehat{\ }\widehat{\ }(Suc\ i))\ (pfirst\ P)) =$
$pleast\ (nnth\ (pleast\text{-}niterates\ P)\ i)\ P$
using *assms*
proof (*induction i*)
case *0*
then show *?case* **by** (*simp add*: *pleast-niterates-first*)
next
case (*Suc i*)
then show *?case*
by (*simp add*: *pleast-niterates-def*)
qed

lemma *holds-nnth-pleast-niterates*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows $P\ (nnth\ (\ pleast\text{-}niterates\ P)\ i)$
using *assms*
proof (*induction i*)
case *0*
then show *?case*
using *LeastI-ex pfirst-def pleast-niterates-first* **by** *auto*
next
case (*Suc i*)
then show *?case*
proof −
have *1*: $P\ (nnth\ (pleast\text{-}niterates\ P)\ (Suc\ i)) = P\ (((\lambda\ s.\ pleast\ s\ P)\widehat{\ }\widehat{\ }(Suc\ i))\ (pfirst\ P))$
using *assms pleast-niterates-nnth* **by** *auto*
have *2*: $(((\lambda\ s.\ pleast\ s\ P)\widehat{\ }\widehat{\ }(Suc\ i))\ (pfirst\ P)) =$
$pleast\ (nnth\ (pleast\text{-}niterates\ P)\ i)\ P$
using *assms pleast-niterates-Suc-pleast* **by** *auto*
have *3*: $P\ (nnth\ (pleast\text{-}niterates\ P)\ i)$
by (*simp add*: *Suc.IH assms*)
have *4*: $P\ (pleast\ (nnth\ (pleast\text{-}niterates\ P)\ i)\ P)$
by (*simp add*: *3 assms holds-pleast-next*)
show *?thesis*
using *1 2 4* **by** *auto*
qed
qed

lemma *nnth-pleast-niterates-mono*:
assumes $\forall kl.\ \exists k{::}nat.\ P\ (kl + k)$
shows $(nnth\ (\ pleast\text{-}niterates\ P)\ i) < (nnth\ (\ pleast\text{-}niterates\ P)\ (Suc\ i))$
using *assms*
proof (*induction i*)
case *0*
then show *?case*
by (*simp add*: *pfirst-less-pleast-pfirst pleast-niterates-nnth*)
next
case (*Suc i*)
then show *?case*

by (*simp add*: *pleast-less-pleast-pleast pleast-niterates-nnth*)
qed

lemma *ndistinct-pleast-niterates*:
  assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
  shows *ndistinct* (*pleast-niterates P*)
using *assms*
by (*simp add*: *nidx-expand nidx-imp-ndistinct nnth-pleast-niterates-mono*)

lemma *not-before-pleast-niterates-zero*:
 assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
        $l{<}\ (nnth\ (\ pleast\text{-}niterates\ P)\ 0)$
 shows $\neg P\ l$
using *assms* **unfolding** *pfirst-def pleast-niterates-def*
using *not-less-Least* **by** *auto*

lemma *not-between-pleast-niterates*:
  assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
          $(nnth\ (\ pleast\text{-}niterates\ P)\ i) < l$
          $l < (nnth\ (\ pleast\text{-}niterates\ P)\ (Suc\ i))$
  shows $\neg\ P\ l$
proof −
 have *1*: $nnth\ (\ pleast\text{-}niterates\ P)\ i = ((\lambda\ s.\ pleast\ s\ P)\widehat{\ }\widehat{\ }i)\ (pfirst\ P)$
   using *assms* **by** (*simp add*: *pleast-niterates-def*)
 have *2*: $(nnth\ (\ pleast\text{-}niterates\ P)\ (Suc\ i)) = pleast\ (nnth\ (pleast\text{-}niterates\ P)\ i)\ P$
   using *assms* **by** (*simp add*: *pleast-niterates-def*)
 have *3*: $P\ (nnth\ (\ pleast\text{-}niterates\ P)\ i)$
   using *assms* **by** (*simp add*: *holds-nnth-pleast-niterates*)
 have *4*: $P\ (nnth\ (\ pleast\text{-}niterates\ P)\ (Suc\ i))$
   using *assms* **by** (*simp add*: *holds-nnth-pleast-niterates*)
 have *5*: $pleast\ (nnth\ (pleast\text{-}niterates\ P)\ i)\ P = (LEAST\ n.\ nnth\ (pleast\text{-}niterates\ P)\ i < n \wedge P\ n)$
   unfolding *pleast-def* **by** *simp*
 show *?thesis*
   using *2 5 assms*(*2*,*3*) *not-less-Least* **by** *auto*
qed

lemma *inf-many-infinite-list-of*:
 assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
 shows $\exists\, nl\ .$ *infinite-list-of P nl*
using *assms* **unfolding** *infinite-list-of-def*
by (*metis holds-nnth-pleast-niterates nfinite-niterates nnth-pleast-niterates-mono not-before-pleast-niterates-zero not-between-pleast-niterates*
  pleast-niterates-def)

lemma *inf-many-w-infinite-list-of*:
 assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
 shows $\exists\, nl\ .$ *w-infinite-list-of P nl*

using *assms* **unfolding** *w-infinite-list-of-def*
by (*metis holds-nnth-pleast-niterates nfinite-niterates nnth-pleast-niterates-mono pleast-niterates-def*)

lemma *holds-pleast-pfirst*:
assumes $\forall\, kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
shows $P\ (pleast\ (pfirst\ P)\ P)$
using *assms* **unfolding** *pleast-def*
pfirst-def
by (*metis* (*no-types*, *lifting*) *LeastI gt-ex trans-less-add1*)

lemma *infinite-list-of-exists*:
assumes *infinite-list-of P nl*
shows $(kl{::}nat) < nnth\ nl\ 0 \vee (\exists\, i.\ kl = nnth\ nl\ i) \vee (\exists\, i.\ nnth\ nl\ i < kl \wedge kl < nnth\ nl\ (Suc\ i))$
using *assms* **unfolding** *infinite-list-of-def*
proof (*induction kl*)
case *0*
then show *?case*
by (*metis bot-nat-0.not-eq-extremum*)
next
case (*Suc kl*)
then show *?case* **apply** *simp*
using *less-Suc-eq* **by** *blast*
qed

lemma *w-infinite-list-of-exists*:
assumes *w-infinite-list-of P nl*
shows $(kl{::}nat) < nnth\ nl\ 0 \vee (\exists\, i.\ kl = nnth\ nl\ i) \vee (\exists\, i.\ nnth\ nl\ i < kl \wedge kl < nnth\ nl\ (Suc\ i))$
using *assms* **unfolding** *w-infinite-list-of-def*
proof (*induction kl*)
case *0*
then show *?case*
by *auto*
next
case (*Suc kl*)
then show *?case* **apply** *simp*
using *less-Suc-eq* **by** *blast*
qed

lemma *infinite-list-of-inf-many*:
assumes $\exists\, nl\ .$ *infinite-list-of P nl*
shows $\forall\ \ kl.\ \exists\, k{::}nat.\ P\ (kl + k)$
proof *auto*
fix *kl*
show $\exists\, k{::}nat.\ P\ (kl + k)$
proof −
obtain *nl* **where** *1*: $\neg nfinite\ nl \wedge$
$(\forall\, i.\ P\ (nnth\ nl\ i)) \wedge$
$(\forall\, i\ .\ nnth\ nl\ i < nnth\ nl\ (Suc\ i)) \wedge$
$(\forall\, l.\ l < nnth\ nl\ 0 \longrightarrow \neg\ P\ l) \wedge$

```
        (∀ i l.  nnth nl i < l ∧ l <  nnth nl (Suc i) ⟶ ¬ P l)
  using assms infinite-list-of-def by auto
 have 2:  kl < nnth nl 0 ⟹ ?thesis
    by (metis 1 less-imp-add-positive)
 have 3:  ⋀ i. kl = nnth nl i ⟹ ?thesis
   using 1 by (metis add.right-neutral)
 have 4:  ⋀ i. nnth nl i < kl ∧ kl < nnth nl (Suc i) ⟹ ?thesis
   using 1 by (metis less-imp-add-positive)
 have 5: kl < nnth nl 0 ∨ (∃ i. kl = nnth nl i) ∨ (∃ i. nnth nl i < kl ∧ kl < nnth nl (Suc i))
   using 1
   proof (induction kl)
   case 0
   then show ?case
   by (metis bot-nat-0.not-eq-extremum)
   next
   case (Suc kl)
   then show ?case apply simp
   using less-Suc-eq by blast
  qed
 show ?thesis
 using 2 3 4 5 by blast
qed
qed

lemma w-infinite-list-of-inf-many:
 assumes ∃ nl . w-infinite-list-of P nl
 shows ∀  kl. ∃ k::nat. P (kl + k)
proof auto
 fix kl
 show  ∃ k::nat. P (kl + k)
 proof  −
 obtain nl where 1: ¬nfinite nl ∧
        (∀ i. P (nnth nl i)) ∧
        (∀ i . nnth nl i < nnth nl (Suc i))
  using assms w-infinite-list-of-def by blast
  have 2:  kl < nnth nl 0 ⟹ ?thesis
    by (metis 1 le-add-diff-inverse order-less-imp-le)
  have 3:  ⋀ i. kl = nnth nl i ⟹ ?thesis
  using 1 by (metis add.right-neutral)
 have 4:  ⋀ i. nnth nl i < kl ∧ kl < nnth nl (Suc i) ⟹ ?thesis
   by (metis 1 less-imp-add-positive)
 have 5: kl < nnth nl 0 ∨ (∃ i. kl = nnth nl i) ∨ (∃ i. nnth nl i < kl ∧ kl < nnth nl (Suc i))
   using 1
   proof (induction kl)
   case 0
   then show ?case
   by auto
   next
   case (Suc kl)
   then show ?case apply simp
```

using *Suc-lessI less-SucI* **by** *blast*
qed
show *?thesis*
using *2 3 4 5* **by** *blast*
qed
qed

lemma *inf-many-eq-infinite-list-of*:
shows $(\forall\ kl.\ \exists\, k{::}nat.\ P\ (kl + k)) \longleftrightarrow (\exists\, nl\ .\ infinite\text{-}list\text{-}of\ P\ nl\ )$
using *inf-many-infinite-list-of*[*of P*] *infinite-list-of-inf-many*[*of P*]
by *blast*

lemma *inf-many-eq-w-infinite-list-of*:
shows $(\forall\ kl.\ \exists\, k{::}nat.\ P\ (kl + k)) \longleftrightarrow (\exists\, nl\ .\ w\text{-}infinite\text{-}list\text{-}of\ P\ nl\ )$
using *inf-many-w-infinite-list-of*[*of P*] *w-infinite-list-of-inf-many*[*of P*]
by *blast*

lemma *holds-eq-exist-one*:
assumes *infinite-list-of P nl*
shows $P\ l \longleftrightarrow (\exists\ n.\ l = (nnth\ nl\ n))$
using *assms infinite-list-of-exists*
unfolding *infinite-list-of-def*
by (*metis*)

lemma *not-holds-eq*:
assumes *infinite-list-of P nl*
shows $\neg\ P\ l \longleftrightarrow$
$(l{::}nat) < nnth\ nl\ 0 \vee (\exists\, i.\ nnth\ nl\ i < l \wedge l < nnth\ nl\ (Suc\ i))$
using *assms infinite-list-of-def* **apply** *simp*
using *infinite-list-of-exists*[*of P nl l*]
by *blast*

lemma *infinite-list-of-mono*:
assumes *infinite-list-of P nl1*
shows $nnth\ nl1\ n < nnth\ nl1\ m \longleftrightarrow n < m$
using *assms* **unfolding** *infinite-list-of-def*
by (*metis lift-Suc-mono-less-iff*)

lemma *w-infinite-list-of-mono*:
assumes *w-infinite-list-of P nl1*
shows $nnth\ nl1\ n < nnth\ nl1\ m \longleftrightarrow n < m$
using *assms* **unfolding** *w-infinite-list-of-def*
using *lift-Suc-mono-less-iff* **by** *blast*

lemma *infinite-list-of-imp-w-infinite-list-of*:
infinite-list-of P nl $\Longrightarrow$ *w-infinite-list-of P nl*
unfolding *infinite-list-of-def w-infinite-list-of-def*
by *auto*

lemma *w-infinite-list-of-and-prev-since*:
assumes ($\exists$ *nl1*. *infinite-list-of P nl1*)
($\exists$ *nl2*. *infinite-list-of Q nl2*)
shows ($\exists$ *nl3*. *w-infinite-list-of* (*and-prev-since Q P*) *nl3*)
proof −
obtain *nl1* **where** *1*: *infinite-list-of P nl1*
using *assms* **by** *blast*
obtain *nl2* **where** *2*: *infinite-list-of Q nl2*
using *assms* **by** *blast*
have *3*: *P* (*nnth nl1 0*)
using *1* **unfolding** *infinite-list-of-def* **by** *blast*
have *4*: (*nnth nl1 0*) < *nnth nl2 0* $\vee$ ($\exists$ *i*. (*nnth nl1 0*) = *nnth nl2 i*) $\vee$
($\exists$ *i*. *nnth nl2 i* < (*nnth nl1 0*) $\wedge$ (*nnth nl1 0*) < *nnth nl2* (*Suc i*))
using *2* **unfolding** *infinite-list-of-def* **using**
infinite-list-of-exists[*of Q nl2*] *2* **by** *presburger*
have *41*: (*nnth nl1 0*) < *nnth nl2 0* $\Longrightarrow$
$\exists$ *j* . (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)
using *1 2* **unfolding** *infinite-list-of-def and-prev-since-def*
by *metis*
have *42*: ($\exists$ *i*. (*nnth nl1 0*) = *nnth nl2 i*) $\Longrightarrow$
$\exists$ *j* . (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)
using *1 2* **unfolding** *infinite-list-of-def and-prev-since-def*
by *metis*
have *43*: ($\exists$ *i*. *nnth nl2 i* < (*nnth nl1 0*) $\wedge$ (*nnth nl1 0*) < *nnth nl2* (*Suc i*)) $\Longrightarrow$
$\exists$ *j* . (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)
using *1 2* **unfolding** *infinite-list-of-def and-prev-since-def*
by (*meson dual-order.strict-trans*)
have *5*: $\exists$ *j* . (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)
using *4 41 42 43* **by** *auto*
have *51*: (*and-prev-since Q P*) (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*)
(*nnth nl2 j*)))
using *5 LeastI-ex*[*of* λ*j*. (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)]
by *auto*
have *6*: $\exists$ *j* . (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*) $\wedge$
($\forall$ *l*. *l* < *j* $\longrightarrow$ $\neg$((*nnth nl2 l*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 l*)))
using *Least-def*[*of* λ*j*. (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)]
51 5 LeastI-ex[*of* λ*j*. (*nnth nl2 j*) > (*nnth nl1 0*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)]
using *not-less-Least* **by** *blast*

have *7*: $\bigwedge$*l*. $\exists$ *j* . (*nnth nl2 j*) > (*nnth nl1 l*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)
proof −
fix *l*
show $\exists$ *j* . (*nnth nl2 j*) > (*nnth nl1 l*) $\wedge$ (*and-prev-since Q P*) (*nnth nl2 j*)
proof −

have *8*: (*nnth nl1 l*) < *nnth nl2 0* ∨ (∃ *i*. (*nnth nl1 l*) = *nnth nl2 i*) ∨
(∃ *i*. *nnth nl2 i* < (*nnth nl1 l*) ∧ (*nnth nl1 l*) < *nnth nl2* (*Suc i*))
using *2* **unfolding** *infinite-list-of-def* **using**
infinite-list-of-exists[*of Q nl2*] *2* **by** *presburger*
show *?thesis*
using *1 2 8* **unfolding** *infinite-list-of-def and-prev-since-def*
apply *auto*
apply *blast*
apply *metis*
by (*meson order-less-trans*)
qed
qed
have *81*: ⋀*l*. (*and-prev-since Q P*) (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 l*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*)))
using *7* **by** (*metis* (*no-types*, *lifting*) *LeastI-ex*)
have *9*: ⋀*l*. ∃ *j* . (*nnth nl2 j*) > (*nnth nl1 l*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*) ∧
(∀ *l1* . *l1* < *j* ⟶ ¬((*nnth nl2 l1*) > (*nnth nl1 l*) ∧ (*and-prev-since Q P*) (*nnth nl2 l1*)))
proof −
fix *l*
show ∃ *j* . (*nnth nl2 j*) > (*nnth nl1 l*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*) ∧
(∀ *l1* . *l1* < *j* ⟶ ¬((*nnth nl2 l1*) > (*nnth nl1 l*) ∧ (*and-prev-since Q P*) (*nnth nl2 l1*)))
using *7* **by** (*metis* (*no-types*, *lifting*) *81 LeastI-ex not-less-Least*)
qed
have *10*: (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*))) < *nnth nl1 0* ∨
(∃ *i*. (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*))) = *nnth nl1 i*) ∨
(∃ *i*. *nnth nl1 i* < (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*))) ∧
(*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*))) < *nnth nl1* (*Suc i*))
using *infinite-list-of-exists*[*of P nl1*] *1* **unfolding** *infinite-list-of-def*
by *blast*

have *11*: ∃ *n*. (*nnth nl1 n*) > (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*))) ∧
P (*nnth nl1 n*)
using *1 2 10* **unfolding** *infinite-list-of-def and-prev-since-def*
apply *auto* **by** *blast*
have *12*: *P* (*nnth nl1* (*LEAST n*. (*nnth nl1 n*) > (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*)))))
using *11 LeastI-ex*[*of* λ*n*. (*nnth nl1 n*) > (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 0*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*)))]
by (*meson 1 infinite-list-of-def*)
have *13*: ⋀*l*. ∃ *n* . (*nnth nl1 n*) > (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 l*) ∧ (*and-prev-since Q P*) (*nnth nl2 j*))) ∧
P (*nnth nl1 n*)
proof −
fix *l*
show ∃ *n* . (*nnth nl1 n*) > (*nnth nl2* (*LEAST j*. (*nnth nl2 j*) > (*nnth nl1 l*) ∧ (*and-prev-since Q P*)

```
(nnth nl2 j))) ∧
          P (nnth nl1 n)
    proof −
     have 14: (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 l) ∧  (and-prev-since Q P) (nnth nl2 j))) <
nnth nl1 l ∨
          (∃ i. (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 l) ∧  (and-prev-since Q P) (nnth nl2 j))) =
nnth nl1 i) ∨
         (∃ i. nnth nl1 i < (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 l) ∧  (and-prev-since Q P) (nnth
nl2 j))) ∧
             (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 l) ∧  (and-prev-since Q P) (nnth nl2 j))) < nnth
nl1 (Suc i))
      using  infinite-list-of-exists[of P nl1] 1 unfolding infinite-list-of-def apply simp
      by (meson linorder-neqE-nat order-less-trans)
     show ?thesis using 1 2 14 unfolding infinite-list-of-def  and-prev-since-def
     apply auto by blast
    qed
   qed
  have 15: ∃  nl3 . ¬ nfinite nl3 ∧
                  nnth nl3 0 = (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 0) ∧  (and-prev-since Q P)
(nnth nl2 j))) ∧
                 (∀  i. nnth nl3 (Suc i) =
                      (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 (LEAST n. (nnth nl1 n) > (nnth nl3
i))) ∧
                                          (and-prev-since Q P) (nnth nl2 j))))

     by auto
  obtain nl3 where 16: ¬ nfinite nl3 ∧
                  nnth nl3 0 = (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 0) ∧  (and-prev-since Q P)
(nnth nl2 j))) ∧
                 (∀  i. nnth nl3 (Suc i) =
                      (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 (LEAST n. (nnth nl1 n) > (nnth nl3
i))) ∧
                                          (and-prev-since Q P) (nnth nl2 j))))
     using 15 by blast
  have 17 :  (∀  i. (and-prev-since Q P) (nnth nl3 i))
    using 1 2 51 13 16 unfolding infinite-list-of-def apply auto
     by (metis (no-types, lifting) 9 LeastI-ex old.nat.exhaust)
  have 18: nnth nl3 0 = (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 0) ∧  (and-prev-since Q P) (nnth
nl2 j)))
   using 16 by auto
  have 19: nnth nl3 1 =
                      (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 (LEAST n. (nnth nl1 n) > (nnth nl3
0))) ∧
                                          (and-prev-since Q P) (nnth nl2 j)))
   using 16 by auto
  have 20: (nnth nl1 (LEAST n. (nnth nl1 n) > (nnth nl3 0))) > nnth nl3 0
   by (metis 11 18 LeastI-ex)

  have 27: ⋀i. nnth nl3 i < nnth nl3 (Suc i)
     proof −
```

```
      fix i
      show nnth nl3 i < nnth nl3 (Suc i)
      proof −
       have 28: (nnth nl1 (LEAST n. (nnth nl1 n) > (nnth nl3 i))) > nnth nl3 i
         proof (cases i)
         case 0
         then show ?thesis
         by (simp add: 20)
         next
         case (Suc nat)
         then show ?thesis
          proof −
            have 280: ⋀ i. ( nnth nl3 (Suc i) =
            nnth nl2 (LEAST j. nnth nl1 (LEAST n. nnth nl3 i < nnth nl1 n) < nnth nl2 j ∧ and-prev-since
Q P (nnth nl2 j)))
               using 16 by blast
            have 281: ( nnth nl3 (Suc nat) =
                  nnth nl2 (LEAST j. nnth nl1 (LEAST n. nnth nl3 nat < nnth nl1 n) < nnth nl2 j ∧
and-prev-since Q P (nnth nl2 j)))
                 using 280 by blast
            show ?thesis using 13 281
            using Suc by (metis LeastI-ex)
          qed
         qed
       have 29: nnth nl3 (Suc i) = (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 (LEAST n. (nnth nl1 n)
> (nnth nl3 i))) ∧
                                        (and-prev-since Q P) (nnth nl2 j)))
        using 16 by blast
       have 30: (nnth nl2 (LEAST j. (nnth nl2 j) > (nnth nl1 (LEAST n. (nnth nl1 n) > (nnth nl3 i))) ∧
                                        (and-prev-since Q P) (nnth nl2 j))) >
           (nnth nl1 (LEAST n. (nnth nl1 n) > (nnth nl3 i)))
        by (metis (no-types, lifting) 7 LeastI-ex)
       show ?thesis
       using 28 29 30 by linarith
      qed
     qed
  show ?thesis unfolding w-infinite-list-of-def
  using 16 17 27 by blast
qed
```

```
lemma inf-many-and-a:
 assumes ∀ kl. ∃ k1 ::nat. P1 (kl+k1)
         ∀ kl. ∃ k2 ::nat. P2 (kl+k2)
 shows   ∀ kl. ∃ k3 ::nat. (and-prev-since P1 P2) (kl+k3)
using assms
w-infinite-list-of-and-prev-since[of P1 P2]
using inf-many-eq-infinite-list-of inf-many-eq-w-infinite-list-of
```

w-infinite-list-of-and-prev-since **by** *auto*

lemma *inf-many-and-b*:
assumes $\forall\, kl.\ \exists\, k3 :: nat.\ (and\text{-}prev\text{-}since\ P1\ P2)\ (kl{+}k3)$
shows $\forall\, kl.\ \exists\, k1 :: nat.\ P1\ (kl{+}k1)$
using *assms* **unfolding** *and-prev-since-def*
by *auto*

lemma *inf-many-and-c*:
assumes $\forall\, kl.\ \exists\, k3 :: nat.\ (and\text{-}prev\text{-}since\ P1\ P2)\ (kl{+}k3)$
shows $\forall\, kl.\ \exists\, k2 :: nat.\ P2\ (kl{+}k2)$
using *assms* **unfolding** *and-prev-since-def*
by (*metis gt-ex le-Suc-ex linorder-le-less-linear trans-less-add1*)

lemma *inf-many-and*:
$(\forall\, kl.\ \exists\, k1 :: nat.\ P1\ (kl{+}k1)) \wedge (\forall\, kl.\ \exists\, k2 :: nat.\ P2\ (kl{+}k2)) \longleftrightarrow$
$(\forall\, kl.\ \exists\, k3 :: nat.\ (and\text{-}prev\text{-}since\ P1\ P2)\ (kl{+}k3))$
using *inf-many-and-a inf-many-and-b inf-many-and-c* **by** *metis*

lemma *inf-many-ge-inf-many-plus*:
$(\forall\, kl.\ \exists\, k2 :: nat.\ kl \leq k2 \wedge P2\ k2) \longleftrightarrow (\forall\, kl.\ \exists\, k2 :: nat.\ P2\ (kl{+}k2))$
by (*metis nat-le-iff-add*)

lemma *inf-many-and-alt*:
$(\forall\, kl.\ \exists\, k1 :: nat.\ kl \leq k1 \wedge P1\ k1) \wedge (\forall\, kl.\ \exists\, k2 :: nat.\ kl \leq k2 \wedge P2\ k2) \longleftrightarrow$
$(\forall\, kl.\ \exists\, k3 :: nat.\ kl \leq k3 \wedge (and\text{-}prev\text{-}since\ P1\ P2)\ k3)$
using *inf-many-and*[*of P1 P2*] *inf-many-ge-inf-many-plus*[*of P1*]
inf-many-ge-inf-many-plus[*of P2*] *inf-many-ge-inf-many-plus*[*of* (*and-prev-since P1 P2*)]
by *presburger*

lemma *inf-many-ntaken-inter*:
assumes *finitary S1*
finitary S2
$\neg$ *nfinite* σ
shows $(\forall\ kl.\ (\exists\ k1.\ kl \leq k1 \wedge (ntaken\ k1\ \sigma) \in S1)) \wedge (\forall\ kl.\ (\exists\ k2.\ kl \leq k2\ \wedge (ntaken\ k2\ \sigma) \in S2)) \longleftrightarrow$
$(\forall\ kl.\ (\exists\ k3.\ kl \leq k3\ \wedge (ntaken\ k3\ \sigma) \in S1 \wedge\ (\exists\ k2.\ k2 < k3\ \wedge ntaken\ k2\ \sigma \in S2 \wedge$
$(\forall\, k22.\ k2 < k22 \wedge k22 < k3\ \longrightarrow ntaken\ k22\ \sigma \notin S1))))$
proof −
have *1*: $((\forall\, kl.\ \exists\, k1{\geq}kl.\ ntaken\ k1\ \sigma \in S1) \wedge (\forall\, kl.\ \exists\, k2{\geq}kl.\ ntaken\ k2\ \sigma \in S2)) =$
$(\forall\, kl.\ \exists\, k3{\geq}kl.\ and\text{-}prev\text{-}since\ (\lambda k1.\ ntaken\ k1\ \sigma \in S1)\ (\lambda k1.\ ntaken\ k1\ \sigma \in S2)\ k3)$
using *inf-many-and-alt*[*of* $(\lambda k1.\ (ntaken\ k1\ \sigma) \in S1)\ (\lambda\ k1.\ (ntaken\ k1\ \sigma) \in S2)$]
by *blast*
have *2*: $(\forall\, kl.\ \exists\, k3{\geq}kl.\ and\text{-}prev\text{-}since\ (\lambda k1.\ ntaken\ k1\ \sigma \in S1)\ (\lambda k1.\ ntaken\ k1\ \sigma \in S2)\ k3) \longleftrightarrow$
$(\forall\ kl.\ (\exists\ k3.\ kl \leq k3\ \wedge (ntaken\ k3\ \sigma) \in S1 \wedge\ (\exists\ k2.\ k2 < k3\ \wedge ntaken\ k2\ \sigma \in S2 \wedge$
$(\forall\, k22.\ k2 < k22 \wedge k22 < k3\ \longrightarrow ntaken\ k22\ \sigma \notin S1))))$
unfolding *and-prev-since-def* **by** *blast*
show *?thesis* **using** *1 2* **by** *simp*
qed

7.8.6 Ap, Ep, Rp and Pp

lemma *in-Ap*:
$\sigma \in Ap\ S \longleftrightarrow (\forall\ \sigma 1.\ (\sigma 1 \preceq \sigma) \longrightarrow \sigma 1 \in S)$
unfolding *Ap-def* **by** *blast*

lemma *in-Ep*:
$\sigma \in Ep\ S \longleftrightarrow (\exists\ \sigma 1.\ (\sigma 1 \preceq \sigma) \wedge \sigma 1 \in S)$
unfolding *Ep-def* **by** *blast*

lemma *in-Ap-alt*:
shows $\sigma \in Ap\ S \longleftrightarrow (\forall k.\ k \leq nlength\ \sigma \longrightarrow (ntaken\ k\ \sigma) \in S)$
unfolding *Ap-def* **apply** *simp*
by (*metis nprefix-eq-exists-ntaken nprefix-ntaken*)

lemma *in-Ep-alt*:
shows $\sigma \in Ep\ S \longleftrightarrow (\exists k.\ k \leq nlength\ \sigma \wedge (ntaken\ k\ \sigma) \in S)$
unfolding *Ep-def*
apply *simp*
by (*metis nprefix-eq-exists-ntaken nprefix-ntaken*)

lemma *in-Ep-wcat*:
assumes *finitary S*
shows $x \in Ep\ S \longleftrightarrow x \in (S \odot Ts)$
using *assms*
unfolding *in-Ep-alt wcat-def cat-def Ts-def Fin-def finitary-def*
apply *auto*
apply (*metis nfuse-def1 nfuse-ntaken-ndropn*)
apply (*metis nprefix-eq-exists-ntaken nprefix-refl*)
by (*metis nlength-nappend nprefix-eq-exists-ntaken nprefix-nappend*)

lemma *Ep-wcat*:
assumes *finitary S*
shows $Ep\ S = (S \odot Ts)$
using *in-Ep-wcat assms* **by** *blast*

lemma *Ep-f-wcat*:
assumes *finitary S*
shows $Ep\text{-}f\ S = S \odot Fin$
using *assms* **unfolding** *Ep-f-def*
Ep-wcat in-Ep-alt wcat-def cat-def Ts-def Fin-def finitary-def
apply *auto*
apply (*metis Suc-ile-eq antisym-conv2 in-Ep-alt nappend-ntaken-ndropn nfinite-nappend ntaken-all*)
using *Ep-def nprefix-refl* **apply** *fastforce*
apply (*metis in-Ep-alt nprefix-eq-exists-ntaken nprefix-nappend*)
by (*simp add*: *nfinite-nappend*)

lemma *Ep-i-cat*:

assumes *finitary S*
shows *Ep-i S = S · Inf*
using *assms* **unfolding** *Ep-i-def*
Ep-wcat in-Ep-alt wcat-def cat-def Ts-def Inf-def finitary-def
apply *auto*
apply (*metis Suc-ile-eq dual-order.order-iff-strict in-Ep-alt*
nappend-ntaken-ndropn nfinite-nappend ntaken-all)
apply (*simp add*: *Ep-wcat finitary-def wcat-def*)
using *Ts-def cat-def* **apply** *blast*
using *nfinite-nappend* **by** *blast*

lemma *in-Rp*:
$\sigma \in Rp\ S \longleftrightarrow (\forall \sigma 1.\ \sigma 1 \preceq \sigma \longrightarrow (\exists \sigma 2.\ \sigma 1 \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \wedge \sigma 2 \in S))$
unfolding *Rp-def* **by** *auto*

lemma *in-Rp-alt-1*:
shows $\sigma \in Rp\ S \longleftrightarrow$
$nfinite\ \sigma \wedge \sigma \in S \vee$
$\neg nfinite\ \sigma \wedge (\exists_\infty\ k.\ (ntaken\ k\ \sigma) \in S)$
proof (*cases nfinite σ*)
case *True*
then show *?thesis* **unfolding** *Rp-def* **using** *nprefix-refl* **apply** *auto*
by (*metis min.orderE nprefix-eq-exists-ntaken nprefix-refl ntaken-all ntaken-nlength*)
next
case *False*
then show *?thesis*
proof −
have *1*: $(\sigma \in Rp\ S) \longleftrightarrow$
$(\forall \sigma 1.\ \sigma 1 \preceq \sigma \longrightarrow (\exists \sigma 2.\ \sigma 1 \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \wedge \sigma 2 \in S))$
unfolding *Rp-def* **by** *simp*
have *2*: $(\forall \sigma 1.\ \sigma 1 \preceq \sigma \longrightarrow (\exists \sigma 2.\ \sigma 1 \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \wedge \sigma 2 \in S)) \longleftrightarrow$
$(\forall k.\ (\exists\ \sigma 2.\ (ntaken\ k\ \sigma) \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \wedge \sigma 2 \in S))$
using *False nprefix-eq-exists-ntaken nprefix-ntaken-infinite*
by *metis*
have *3*: $(\forall k.\ (\exists\ \sigma 2.\ (ntaken\ k\ \sigma) \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \wedge \sigma 2 \in S)) \longleftrightarrow$
$(\forall k.\ (\exists\ m.\ k \leq m \wedge (ntaken\ k\ \sigma) \preceq ntaken\ m\ \sigma \wedge ntaken\ m\ \sigma \preceq \sigma \wedge ntaken\ m\ \sigma \in S))$
unfolding *nprefix-eq-exists-ntaken* **using** *False* **apply** *auto*
apply (*metis enat-ord-simps(1) min.cobounded1 min.commute min-def ntaken-all ntaken-nlength*)
by (*metis enat-ord-simps(1,3) enat-the-enat min.orderE nfinite-conv-nlength-enat ntaken-nlength*
ntaken-ntaken)
have *4*: $(\forall k.\ (\exists\ m.\ k \leq m \wedge (ntaken\ k\ \sigma) \preceq ntaken\ m\ \sigma \wedge ntaken\ m\ \sigma \preceq \sigma \wedge ntaken\ m\ \sigma \in S)) \longleftrightarrow$
$(\forall m.\ \exists n{\geq}m.\ ntaken\ n\ \sigma \in S)$
apply *auto*
by (*metis False nfinite-ntaken nle-le ntaken-all*
nprefix-ntaken nprefix-ntaken-ntaken-alt)
have *10*: $(\exists_\infty\ k.\ (ntaken\ k\ \sigma) \in S) \longleftrightarrow (\forall m.\ \exists n{\geq}m.\ ntaken\ n\ \sigma \in S)$
using *INFM-nat-le*[*of* $(\lambda\ k.\ (ntaken\ k\ \sigma) \in S)$] **by** *blast*
show *?thesis*
using *1 10 2 3 4 False* **by** *presburger*

qed
qed

lemma *in-Rp-alt-2*:
shows $\sigma \in Rp\ S \longleftrightarrow$
$nfinite\ \sigma \wedge \sigma \in S \vee$
$\neg nfinite\ \sigma \wedge (\forall\, m.\ (\exists\ n \geq m.\ (ntaken\ n\ \sigma) \in S))$
using *in-Rp-alt-1*[*of* σ S] *INFM-nat-le*[*of* $(\lambda\ k.\ (ntaken\ k\ \sigma) \in S)$]
by *blast*

lemma *in-Pp*:
shows $\sigma \in Pp\ S \longleftrightarrow (\exists \sigma 1.\ \sigma 1 \preceq \sigma \wedge (\forall \sigma 2.\ \sigma 1 \preceq \sigma 2 \wedge \sigma 2 \preceq \sigma \longrightarrow \sigma 2 \in S))$
unfolding *Pp-def* **by** *auto*

lemma *in-Pp-alt-1*:
shows $\sigma \in Pp\ S \longleftrightarrow$
$nfinite\ \sigma \wedge \sigma \in S \vee$
$\neg nfinite\ \sigma \wedge (\forall_\infty\ k.\ (ntaken\ k\ \sigma) \in S)$
proof −
have *1*: $\sigma \in Pp\ S \longleftrightarrow \sigma \notin Rp\ (Fin\ -S)$
unfolding *Pp-def Rp-def* **apply** *simp*
by (*metis Fin-def mem-Collect-eq nprefix-eq-exists-ntaken*)
have *2*: $\sigma \notin Rp\ (Fin\ -S) \longleftrightarrow$
$\neg(nfinite\ \sigma \wedge \sigma \in (Fin\ -S) \vee$
$\neg nfinite\ \sigma \wedge (\exists_\infty\ k.\ (ntaken\ k\ \sigma) \in (Fin\ -S)))$
by (*simp add*: *in-Rp-alt-1*)
have *3*: $\neg(nfinite\ \sigma \wedge \sigma \in (Fin\ -S) \vee$
$\neg nfinite\ \sigma \wedge (\exists_\infty\ k.\ (ntaken\ k\ \sigma) \in (Fin\ -S))) \longleftrightarrow$
$nfinite\ \sigma \wedge \sigma \in S \vee$
$\neg nfinite\ \sigma \wedge (\forall_\infty\ k.\ (ntaken\ k\ \sigma) \in S)$
by (*metis* (*mono-tags*, *lifting*) *DiffD2 DiffI Fin-def INFM-nat-le MOST-nat-le*
mem-Collect-eq nfinite-ntaken)
show *?thesis*
using *1 2 3* **by** *order*
qed

lemma *in-Pp-alt-2*:
shows $\sigma \in Pp\ S \longleftrightarrow$
$nfinite\ \sigma \wedge \sigma \in S \vee$
$\neg nfinite\ \sigma \wedge (\exists\ m.\ (\forall\ n \geq m.\ (ntaken\ n\ \sigma) \in S))$
using *in-Pp-alt-1*[*of* σ S] *MOST-nat-le*[*of* $(\lambda\ k.\ (ntaken\ k\ \sigma) \in S)$] **by** *blast*

lemma *Ap-i-in-eqv-Ep-i*:
assumes $S \subseteq Fin$
shows $x \in Inf\ -\ Ap\text{-}i\ S \longleftrightarrow\ x \in Ep\text{-}i(Fin\ -\ S)$
unfolding *Fin-def Ap-i-def Ep-i-def Inf-def Ap-def Ep-def*
apply *auto*

using *nprefix-def nprefix-eq-def* **by** *blast*

lemma *Ap-i-eqv-Ep-i*:
assumes $S \subseteq Fin$
shows $Inf - Ap\text{-}i\ S = Ep\text{-}i(Fin - S)$
using *assms Ap-i-in-eqv-Ep-i*[*of S*]
by *blast*

lemma *Ep-i-in-eqv-Ap-i*:
assumes $S \subseteq Fin$
shows $x \in Inf - Ep\text{-}i\ S \longleftrightarrow x \in Ap\text{-}i(Fin - S)$
unfolding *Fin-def Ap-i-def Ep-i-def Inf-def Ap-def Ep-def*
apply *auto*
using *nprefix-def nprefix-eq-def* **by** *blast*

lemma *Ep-i-eqv-Ap-i*:
assumes $S \subseteq Fin$
shows $Inf - Ep\text{-}i\ S = Ap\text{-}i(Fin - S)$
using *assms Ep-i-in-eqv-Ap-i*[*of S*] **by** *blast*

lemma *Rp-i-in-eqv-Pp-i*:
assumes $S \subseteq Fin$
shows $x \in Inf - Rp\text{-}i\ S \longleftrightarrow x \in Pp\text{-}i(Fin - S)$
unfolding *Fin-def Rp-i-def Pp-i-def Inf-def Rp-def Pp-def*
apply *auto*
using *nprefix-def nprefix-eq-def* **by** *blast*

lemma *Rp-i-eqv-Pp-i*:
assumes $S \subseteq Fin$
shows $Inf - Rp\text{-}i\ S = Pp\text{-}i(Fin - S)$
using *assms Rp-i-in-eqv-Pp-i*[*of S*]
by *blast*

lemma *Pp-i-in-eqv-Rp-i*:
assumes $S \subseteq Fin$
shows $x \in Inf - Pp\text{-}i\ S \longleftrightarrow x \in Rp\text{-}i(Fin - S)$
unfolding *Fin-def Rp-i-def Pp-i-def Inf-def Rp-def Pp-def*
apply *auto*
using *nprefix-def nprefix-eq-def* **by** *blast*

lemma *Pp-i-eqv-Rp-i*:
assumes $S \subseteq Fin$
shows $Inf - Pp\text{-}i\ S = Rp\text{-}i(Fin - S)$
using *assms Pp-i-in-eqv-Rp-i*[*of S*] **by** *blast*

lemma *Ap-i-inter*:
$Ap\text{-}i\ (S1 \cap S2) = Ap\text{-}i\ S1 \cap Ap\text{-}i\ S2$
unfolding *Ap-i-def Ap-def Inf-def*

by *auto*

lemma *in-Ap-i-union*:
$x\in$ *Ap-i S1* $\cup$ *Ap-i S2* $\longleftrightarrow x \in$ *Ap-i*(*Ap-f S1* $\cup$ *Ap-f S2*)
proof −
have *1*: $x \in$ *Ap-i S1* $\longleftrightarrow (\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$ *ntaken* $k\ x \in S1)$
$\wedge \neg$ *nfinite* x
using *in-Ap-alt*[*of* x *S1*] *Inf-def*
using *Ap-i-def* **by** *auto*
have *2*: $x \in$ *Ap-i S2* $\longleftrightarrow (\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$ *ntaken* $k\ x \in S2)$
$\wedge \neg$ *nfinite* x
using *in-Ap-alt*[*of* x *S2*] *Inf-def*
using *Ap-i-def* **by** *auto*
have *3*: $x \in$ *Ap-i*(*Ap-f S1* $\cup$ *Ap-f S2*) $\longleftrightarrow$
$(\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$ *ntaken* $k\ x \in$ *Ap S1* $\cap$ *Fin* $\cup$ *Ap S2* $\cap$ *Fin*)
$\wedge \neg$ *nfinite* x
using *in-Ap-alt*[*of* x *Ap S1* $\cap$ *Fin* $\cup$ *Ap S2* $\cap$ *Fin*] **using** *Ap-i-def*
by (*metis Ap-f-def Inf-def Int-iff mem-Collect-eq*)
have *4*: $(\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$ *ntaken* $k\ x \in$ *Ap S1* $\cap$ *Fin* $\cup$ *Ap S2* $\cap$ *Fin*)
$\longleftrightarrow$
$(\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$
$(\forall ka.$ *enat* $ka \leq$ *nlength* (*ntaken* $k\ x$) $\longrightarrow$ *ntaken* ka (*ntaken* $k\ x$) $\in S1$)
$\vee$
$(\forall ka.$ *enat* $ka \leq$ *nlength* (*ntaken* $k\ x$) $\longrightarrow$ *ntaken* ka (*ntaken* $k\ x$) $\in S2$))
using *in-Ap-alt*[*of* - *S1*]
using *in-Ap-alt*[*of* - *S2*] **by** (*simp add*: *Fin-def*)
have *5*: $(\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$
$(\forall ka.$ *enat* $ka \leq$ *nlength* (*ntaken* $k\ x$) $\longrightarrow$ *ntaken* ka (*ntaken* $k\ x$) $\in S1$)
$\vee$
$(\forall ka.$ *enat* $ka \leq$ *nlength* (*ntaken* $k\ x$) $\longrightarrow$ *ntaken* ka (*ntaken* $k\ x$) $\in S2$)) $\longleftrightarrow$
$(\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$ *ntaken* $k\ x \in S1$) $\vee$
$(\forall k.$ *enat* $k \leq$ *nlength* $x \longrightarrow$ *ntaken* $k\ x \in S2$)
apply (*auto simp add*: *order-subst2*)
by (*meson linorder-linear*)
show *?thesis*
using *1 2 3 4 5* **by** *force*
qed

lemma *Ap-i-union*:
Ap-i S1 $\cup$ *Ap-i S2* = *Ap-i*(*Ap-f S1* $\cup$ *Ap-f S2*)
using *in-Ap-i-union*[*of* - *S1 S2*] **by** *blast*

lemma *in-Ap-inter*:
$x \in$ *Ap-i S1* $\cap$ *Ap-i S2* $\longleftrightarrow x \in$ *Ap-i* (*S1* $\cap$ *S2*)
unfolding *Ap-i-def* **using** *in-Ap-alt* **apply** *simp*
unfolding *Inf-def*
by (*meson Int-iff in-Ap-alt*)

lemma *Ap-inter*:
Ap-i S1 ∩ Ap-i S2 = Ap-i (S1 ∩ S2)
using *in-Ap-inter* **by** *blast*

lemma *in-Ep-i-inter*:
x∈ Ep-i S1 ∩ Ep-i S2 ⟷ x ∈ Ep-i(Ep-f S1 ∩ Ep-f S2)
proof −
have *1*: *x ∈ Ep-i S1 ⟷ (∃ k. enat k ≤ nlength x ∧ ntaken k x ∈ S1)*
∧ ¬ nfinite x
using *in-Ep-alt*[*of x S1*] *Inf-def*
using *Ep-i-def* **by** *auto*
have *2*: *x ∈ Ep-i S2 ⟷ (∃ k. enat k ≤ nlength x ∧ ntaken k x ∈ S2)*
∧ ¬ nfinite x
using *in-Ep-alt*[*of x S2*] *Inf-def*
using *-Ep-i-def* **by** *auto*
have *3*: *x ∈ Ep-i(Ep-f S1 ∩ Ep-f S2) ⟷*
(∃ k. enat k ≤ nlength x ∧ ntaken k x ∈ Ep S1 ∩ Fin ∩ Ep S2 ∩ Fin)
∧ ¬ nfinite x
using *in-Ep-alt*[*of x Ep S1 ∩ Fin ∩ Ep S2 ∩ Fin*] **using** *Ep-i-def*
by (*metis Ep-f-def Inf-def Int-iff inf-assoc*
mem-Collect-eq)
have *4*: *(∃ k. enat k ≤ nlength x ∧ ntaken k x ∈ Ep S1 ∩ Fin ∩ Ep S2 ∩ Fin)*
⟷
(∃ k. enat k ≤ nlength x ∧
(∃ ka. enat ka ≤ nlength (ntaken k x) ∧ ntaken ka (ntaken k x) ∈ S1)
∧
(∃ ka. enat ka ≤ nlength (ntaken k x) ∧ ntaken ka (ntaken k x) ∈ S2))
using *in-Ep-alt*[*of - S1*]
using *in-Ep-alt*[*of - S2*] **by** (*simp add*: *Fin-def*)
have *5*: *(∃ k. enat k ≤ nlength x ∧*
(∃ ka. enat ka ≤ nlength (ntaken k x) ∧ ntaken ka (ntaken k x) ∈ S1)
∧
(∃ ka. enat ka ≤ nlength (ntaken k x) ∧ ntaken ka (ntaken k x) ∈ S2)) ⟷
(∃ k. enat k ≤ nlength x ∧ ntaken k x ∈ S1) ∧
(∃ k. enat k ≤ nlength x ∧ ntaken k x ∈ S2)
apply (*auto simp add*: *order-subst2*)
by (*metis min.absorb1 nle-le*)
show *?thesis*
using *1 2 3 4 5* **by** *force*
qed

lemma *Ep-i-inter*:
Ep-i S1 ∩ Ep-i S2 = Ep-i(Ep-f S1 ∩ Ep-f S2)
using *in-Ep-i-inter* **by** *blast*

lemma *Ep-union*:
Ep-i S1 ∪ Ep-i S2 = Ep-i (S1 ∪ S2)
unfolding *Ep-def Ep-i-def* **by** *auto*

lemma *in-Rp-i-union*:
assumes *finitary S1*
finitary S2
shows $\sigma\in$ *Rp-i S1* $\cup$ *Rp-i S2* $\longleftrightarrow \sigma \in$ *Rp-i* (*S1* $\cup$ *S2*)
using *assms*
unfolding *Rp-i-def Inf-def Rp-def*
apply *auto*
using *nprefix-eq-exists-ntaken*[*of* - σ]
nprefix-ntaken-ntaken-alt nprefix-trans
by (*metis nle-le*)

lemma *Rp-i-union*:
assumes *finitary S1*
finitary S2
shows *Rp-i S1* $\cup$ *Rp-i S2* $=$ *Rp-i* (*S1* $\cup$ *S2*)
using *assms in-Rp-i-union* **by** *blast*

lemma *in-Rp-i-inf-many*:
assumes *finitary S*
shows $\sigma\in$ *Rp-i S* $\longleftrightarrow$
$\neg$*nfinite* $\sigma \wedge (\forall\ kl.\ (\exists\ k1.\ kl \leq k1 \wedge (ntaken\ k1\ \sigma) \in S))$
using *assms* **unfolding** *finitary-def*
in-Rp-alt-2[*of* σ *S*]
by (*metis Inf-def Int-iff Rp-i-def* ‹($\sigma \in Rp\ S$) = (*nfinite* $\sigma \wedge \sigma \in S \vee \neg$ *nfinite* $\sigma \wedge (\forall m.\ \exists n{\geq}m.$ *ntaken n* $\sigma \in S$))› *mem-Collect-eq*)

lemma *in-Rp-i-inter*:
assumes *finitary S1*
finitary S2
shows $\sigma\in$ *Rp-i S1* $\cap$ *Rp-i S2* $\longleftrightarrow \sigma \in$ *Rp-i*(*S1* $\cap$ (((*Fin* $-$ *S1*) $\mathcal{S}$ *S2*) $\cdot$ *Empty*))
proof $-$
have *1*: $\sigma\in$ *Rp-i S1* $\cap$ *Rp-i S2* $\longleftrightarrow$
$\neg$*nfinite* $\sigma \wedge$
$(\forall\ kl.\ (\exists\ k1.\ kl \leq k1 \wedge (ntaken\ k1\ \sigma) \in S1)) \wedge$
$(\forall\ kl.\ (\exists\ k2.\ kl \leq k2 \wedge (ntaken\ k2\ \sigma) \in S2))$

using *in-Rp-i-inf-many*[*of S1*] *in-Rp-i-inf-many*[*of S2*] *assms*(*1,2*) **by** *auto*
have *2*: $\sigma \in$ *Rp-i*(*S1* $\cap$ (((*Fin* $-$ *S1*) $\mathcal{S}$ *S2*) $\cdot$ *Empty*)) $\longleftrightarrow$
($\neg$ *nfinite* $\sigma \wedge (\forall kl.\ \exists k1{\geq}kl.$ *ntaken k1* $\sigma \in$ *S1* $\cap$ (((*Fin* $-$ *S1*) $\mathcal{S}$ *S2*) $\cdot$ *Empty*)))

using *in-Rp-i-inf-many*[*of S1* $\cap$ (((*Fin* $-$ *S1*) $\mathcal{S}$ *S2*) $\cdot$ *Empty*) σ]
by (*metis IntE assms*(*1*) *finitary-def*)
have *3*: $\bigwedge$ *kl k3*. *kl* $\leq$ *k3* $\wedge \neg$ *nfinite* $\sigma \Longrightarrow$
(*ntaken k3* σ) $\in$ ((*Fin* $-$ *S1*) $\mathcal{S}$ *S2*) $\cdot$ *Empty* $\longleftrightarrow$
$(\exists\ k2.\ k2 < k3\ \wedge$ *ntaken k2* $\sigma \in$ *S2* $\wedge$
$(\forall k22.\ \ k2 < k22 \wedge k22 < k3\ \longrightarrow$ *ntaken k22* $\sigma \notin$ *S1*))

using *assms in-prev-since*[*of S1 S2 σ*] **by** *blast*
have *4*: (¬ *nfinite σ* ∧ (∀ *kl*. ∃ *k1*≥*kl*. *ntaken k1 σ* ∈ *S1* ∩ (((*Fin* − *S1*) $\mathcal{S}$ *S2*) · *Empty*))) ⟷
(¬ *nfinite σ* ∧ (∀ *kl*. ∃ *k1*≥*kl*. *ntaken k1 σ* ∈ *S1* ∧ *ntaken k1 σ* ∈ ((*Fin* − *S1*) $\mathcal{S}$ *S2*) · *Empty*))
by *simp*
have *5*: (¬ *nfinite σ* ∧ (∀ *kl*. ∃ *k1*≥*kl*. *ntaken k1 σ* ∈ *S1* ∧ *ntaken k1 σ* ∈ ((*Fin* − *S1*) $\mathcal{S}$ *S2*) · *Empty*))
⟷ ¬ *nfinite σ* ∧
(∀ *kl*. (∃ *k3*. *kl* ≤ *k3* ∧ (*ntaken k3 σ*) ∈ *S1* ∧ (∃ *k2*. *k2* < *k3* ∧ *ntaken k2 σ* ∈ *S2* ∧
(∀ *k22*. *k2* < *k22* ∧ *k22* < *k3* ⟶ *ntaken k22 σ* ∉ *S1*))))
using *assms* **unfolding** *finitary-def* **using** *3* **apply** *auto* **apply** *meson* **by** *meson*
show *?thesis* **using** *1 5 inf-many-ntaken-inter*[*of S1 S2 σ*]
using *2 4 assms*(*1*,*2*) **by** *argo*
qed

lemma *Rp-i-inter*:
assumes *finitary S1*
finitary S2
shows *Rp-i S1* ∩ *Rp-i S2* = *Rp-i*(*S1* ∩ (((*Fin* − *S1*) $\mathcal{S}$ *S2*) · *Empty*))
using *assms in-Rp-i-inter* **by** *blast*

lemma *Pp-i-inter*:
assumes *finitary S1*
finitary S2
shows *Pp-i S1* ∩ *Pp-i S2* = *Pp-i* (*S1* ∩ *S2*)
proof −
have *1*: *Pp-i S1* = *Inf* − *Rp-i* (*Fin* −*S1*)
using *assms*
by (*metis Pp-i-def Pp-i-eqv-Rp-i double-diff inf-le2 subset-refl subseteq-Fin-eq-finitary*)
have *2*: *Pp-i S2* = *Inf* − *Rp-i* (*Fin* −*S2*)
using *assms*
by (*metis Pp-i-def Pp-i-eqv-Rp-i double-diff inf-le2 subset-refl subseteq-Fin-eq-finitary*)
have *3*: (*Inf* − *Rp-i* (*Fin* −*S1*)) ∩ (*Inf* − *Rp-i* (*Fin* −*S2*)) =
(*Inf* − (*Rp-i* (*Fin* − *S1*) ∪ *Rp-i* (*Fin* − *S2*)))
using *assms* **by** *blast*
have *4*: (*Rp-i* (*Fin* − *S1*) ∪ *Rp-i* (*Fin* − *S2*)) =
Rp-i ((*Fin* − *S1*) ∪ (*Fin* − *S2*))
using *assms* **by** (*simp add*: *Rp-i-union finitary-Fin-complement*)
have *5*: ((*Fin* − *S1*) ∪ (*Fin* − *S2*)) = (*Fin* − (*S1* ∩ *S2*))
using *assms* **by** (*simp add*: *Diff-Int*)
have *6*: (*Inf* − (*Rp-i* (*Fin* − *S1*) ∪ *Rp-i* (*Fin* − *S2*))) =
(*Inf* − *Rp-i* (*Fin* − (*S1* ∩ *S2*)))
by (*simp add*: *4 5*)
have *7*: (*Inf* − *Rp-i* (*Fin* − (*S1* ∩ *S2*))) = *Pp-i* (*S1* ∩ *S2*)
using *assms*
by (*metis Diff-Diff-Int Rp-i-eqv-Pp-i inf.absorb-iff2 inf.coboundedI1 subseteq-Fin-eq-finitary*)
show *?thesis*
by (*simp add*: *1 2 3 4 5 7*)
qed

lemma *Pp-i-union*:
assumes *finitary S1*

finitary S2
shows *Pp-i S1 ∪ Pp-i S2 = Pp-i (S1 ∪ (Fin − ((S1 𝒮 (Fin − S2)) · Empty)))*
proof −
have *1*: *Pp-i S1 = Inf − Rp-i (Fin −S1)*
using *assms*
by (*metis Pp-i-def Pp-i-eqv-Rp-i double-diff inf-le2 subset-refl subseteq-Fin-eq-finitary*)
have *2*: *Pp-i S2 = Inf − Rp-i (Fin −S2)*
using *assms*
by (*metis Pp-i-def Pp-i-eqv-Rp-i double-diff inf-le2 subset-refl subseteq-Fin-eq-finitary*)
have *3*: *(Inf − Rp-i (Fin −S1)) ∪ (Inf − Rp-i (Fin −S2)) =*
(Inf − (Rp-i (Fin − S1) ∩ Rp-i (Fin − S2)))
using *assms* **by** *blast*
have *4*: *(Rp-i (Fin − S1) ∩ Rp-i (Fin − S2)) =*
Rp-i ((Fin − S1) ∩ ((S1 𝒮 (Fin − S2)) · Empty))
using *assms*
by (*metis Diff-Diff-Int Rp-i-inter inf.absorb-iff2 subseteq-Fin-eq-finitary*)
have *5*: *((Fin − S1) ∩ ((S1 𝒮 (Fin − S2)) · Empty)) =*
(Fin − (S1 ∪ (Fin − ((S1 𝒮 (Fin − S2)) · Empty))))
using *assms* **by** *blast*
have *6*: *(Inf − (Rp-i (Fin − S1) ∩ Rp-i (Fin − S2))) =*
(Inf − (Rp-i (Fin − (S1 ∪ (Fin − ((S1 𝒮 (Fin − S2)) · Empty))))))
by (*simp add*: *4 5*)
have *7*: *(Inf − (Rp-i (Fin − (S1 ∪ (Fin − ((S1 𝒮 (Fin − S2)) · Empty)))))) =*
Pp-i (S1 ∪ (Fin − ((S1 𝒮 (Fin − S2)) · Empty)))
using *assms*
by (*metis Diff-Diff-Int Diff-subset Rp-i-eqv-Pp-i inf.absorb-iff2 subseteq-Fin-eq-finitary sup.boundedI*)
show *?thesis*
by (*simp add*: *1 2 3 4 5 7*)
qed

lemma *Ap-i-Fs*:
Ap-i Fs = Fs
unfolding *Ap-i-def Ap-def Fs-def* **apply** *auto*
by (*meson nprefix-eq-def nprefix-ntaken-infinite*)

lemma *Ep-i-Fs*:
Ep-i Fs = Fs
unfolding *Ep-i-def Ep-def Fs-def* **by** *auto*

lemma *Rp-i-Fs*:
Rp-i Fs = Fs
unfolding *Rp-i-def Rp-def Fs-def* **apply** *auto*
by (*meson nprefix-eq-def nprefix-ntaken-infinite*)

lemma *Pp-i-Fs*:
Pp-i Fs = Fs
unfolding *Pp-i-def Pp-def Fs-def* **apply** *auto*
using *nprefix-eq-exists-ntaken nprefix-refl* **by** *auto*

lemma *Ap-i-Ts*:
Ap-i Ts = Inf
unfolding *Ap-i-def Ap-def Ts-def Inf-def* **by** *auto*

lemma *Rp-i-Ts*:
Rp-i Ts = Inf
unfolding *Rp-i-def Rp-def Ts-def Inf-def* **apply** *auto*
using *nprefix-eq-exists-ntaken nprefix-refl* **by** *auto*

lemma *Ap-i-Fin*:
Ap-i Fin = Inf
unfolding *Ap-i-def Ap-def Fin-def Inf-def* **apply** *auto*
using *nprefix-eq-exists-ntaken* **by** *blast*

lemma *Rp-i-Fin*:
Rp-i Fin = Inf
unfolding *Rp-i-def Rp-def Fin-def Inf-def* **apply** *auto*
using *nprefix-eq-exists-ntaken nprefix-refl* **by** *blast*

lemma *Ep-i-Ts*:
Ep-i Ts = Inf
unfolding *Ep-i-def Ep-def Ts-def Inf-def* **apply** *auto*
using *nprefix-ntaken-infinite* **by** *auto*

lemma *Pp-i-Ts*:
Pp-i Ts = Inf
unfolding *Pp-i-def Pp-def Ts-def Inf-def* **apply** *auto*
using *nprefix-ntaken-infinite* **by** *auto*

lemma *Ep-i-Fin*:
Ep-i Fin = Inf
unfolding *Ep-i-def Ep-def Fin-def Inf-def* **apply** *auto*
using *nprefix-eq-exists-ntaken nprefix-ntaken-infinite* **by** *blast*

lemma *Pp-i-Fin*:
Pp-i Fin = Inf
unfolding *Pp-i-def Pp-def Fin-def Inf-def* **apply** *auto*
using *nprefix-eq-exists-ntaken nprefix-ntaken-infinite* **by** *blast*

lemma *Ap-f-Fin*:
Ap-f Fin = Fin
unfolding *Ap-f-def Ap-def Fin-def* **apply** *auto*
by (*simp add*: *nprefix-eq-exists-ntaken*)

lemma *Ep-f-Fin*:
Ep-f Fin = Fin

unfolding *Ep-f-def Ep-def Fin-def* **apply** *auto*
using *nprefix-refl* **by** *blast*

lemma *Ap-i-inter-Inf*:
$(\ Ap\text{-}i\ S \cap Inf) = Ap\text{-}i\ S$
by (*simp add*: *Ap-i-def*)

lemma *Rp-i-inter-Inf*:
$(Rp\text{-}i\ S \cap Inf) = Rp\text{-}i\ S$
by (*simp add*: *Rp-i-def*)

lemma *Ep-i-inter-Inf*:
$(\ Ep\text{-}i\ S \cap Inf) = Ep\text{-}i\ S$
by (*simp add*: *Ep-i-def*)

lemma *Pp-i-inter-Inf*:
$(Pp\text{-}i\ S \cap Inf) = Pp\text{-}i\ S$
by (*simp add*: *Pp-i-def*)

lemma *subset-eq-Ep-f*:
assumes *finitary S*
shows $S \subseteq Ep\text{-}f\ S$
using *assms*
by (*simp add*: *Ep-f-wcat wcat-def*)

7.8.7 safety and guarantee

lemma *safety-guarantee-prop-complementation*:
assumes $S \subseteq Inf$
shows $safety\text{-}prop\ S \longleftrightarrow guarantee\text{-}prop\ (Inf - S)$
using *assms*
unfolding *safety-prop-def guarantee-prop-def*
by (*metis Ap-i-eqv-Ep-i Diff-subset Ep-i-eqv-Ap-i double-diff equalityE subseteq-Fin-eq-finitary subseteq-Inf-eq-infinita*

lemma *guarantee-safety-prop-complementation*:
assumes $S \subseteq Inf$
shows $safety\text{-}prop\ (Inf - S) \longleftrightarrow guarantee\text{-}prop\ S$
using *assms*
by (*simp add*: *double-diff safety-guarantee-prop-complementation*)

lemma *safety-prop-inter*:
assumes *safety-prop S1*
safety-prop S2
shows $safety\text{-}prop\ (S1 \cap S2)$
using *assms* **unfolding** *safety-prop-def*
by (*metis Ap-i-inter le-infI1 subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary*)

lemma *safety-prop-union*:
assumes *safety-prop S1*
safety-prop S2
shows *safety-prop* (*S1* ∪ *S2*)
using *assms* **unfolding** *safety-prop-def*
by (*metis Ap-i-Fin Ap-i-inter Ap-i-inter-Inf Ap-i-union Diff-Diff-Int Diff-subset finitary-Fin-complement inf-commute*
subseteq-Inf-eq-infinitary)

lemma *guarantee-prop-inter*:
assumes *guarantee-prop S1*
guarantee-prop S2
shows *guarantee-prop* (*S1* ∩ *S2*)
using *assms* **unfolding** *guarantee-prop-def*
by (*metis Ep-f-def Ep-i-inter inf-le1 le-infI1 le-inf-iff subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary*)

lemma *guarantee-prop-union*:
assumes *guarantee-prop S1*
guarantee-prop S2
shows *guarantee-prop* (*S1* ∪ *S2*)
using *assms* **unfolding** *guarantee-prop-def*
by (*meson Ep-union subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary sup.boundedI*)

7.8.8 response and persistence

lemma *response-persistence-prop-complementation*:
assumes *S* ⊆ *Inf*
shows *response-prop S* ⟷ *persistence-prop* (*Inf* − *S*)
using *assms*
unfolding *response-prop-def persistence-prop-def*
by (*metis Diff-Diff-Int Pp-i-eqv-Rp-i Rp-i-eqv-Pp-i inf.absorb-iff2 subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary*)

lemma *persistence-response-prop-complementation*:
assumes *S* ⊆ *Inf*
shows *persistence-prop S* ⟷ *response-prop* (*Inf* − *S*)
using *assms*
by (*simp add*: *double-diff response-persistence-prop-complementation*)

lemma *response-prop-inter*:
assumes *response-prop S1*
response-prop S2
shows *response-prop* (*S1* ∩ *S2*)
using *assms* **unfolding** *response-prop-def*
by (*meson Rp-i-inter le-infI1 subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary*)

lemma *response-prop-union*:
assumes *response-prop S1*
response-prop S2

shows *response-prop* (*S1* ∪ *S2*)
using *assms* **unfolding** *response-prop-def*
by (*meson Rp-i-union subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary sup.boundedI*)

lemma *persistence-prop-inter*:
assumes *persistence-prop S1*
persistence-prop S2
shows *persistence-prop* (*S1* ∩ *S2*)
using *assms* **unfolding** *persistence-prop-def*
by (*meson Pp-i-inter le-infI1 subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary*)

lemma *persistence-prop-union*:
assumes *persistence-prop S1*
persistence-prop S2
shows *persistence-prop* (*S1* ∪ *S2*)
using *assms* **unfolding** *persistence-prop-def*
by (*meson Diff-subset Pp-i-union subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary sup.boundedI*)

lemma *in-safety-in-response*:
assumes *finitary S*
shows *σ* ∈ *Ap-i S* ⟷ *σ* ∈ *Rp-i* (*Ap-f S*)
using *assms*
unfolding *Ap-f-def Ap-i-def finitary-def Inf-def Ap-def Rp-i-def Rp-def Fin-def* **apply** *auto*
by (*meson nprefix-eq-exists-ntaken nprefix-refl nprefix-trans*)

lemma *in-guarantee-in-response*:
assumes *finitary S*
shows *σ* ∈ *Ep-i S* ⟷ *σ* ∈ *Rp-i* (*Ep-f S*)
using *assms*
unfolding *Ep-f-def Ep-i-def finitary-def Inf-def Ep-def Rp-i-def Rp-def Fin-def* **apply** *auto*
apply (*metis nle-le nprefix-eq-exists-ntaken nprefix-ntaken-ntaken-alt*)
by (*metis nprefix-ntaken-infinite nprefix-trans*)

lemma *in-safety-in-persistence*:
assumes *finitary S*
shows *σ* ∈ *Ap-i S* ⟷ *σ* ∈ *Pp-i* (*Ap-f S*)
using *assms*
unfolding *Ap-f-def Ap-i-def finitary-def Inf-def Ap-def Pp-i-def Pp-def Fin-def* **apply** *auto*
apply (*metis nprefix-eq-exists-ntaken nprefix-ntaken-infinite nprefix-trans*)
by (*metis nle-le nprefix-eq-exists-ntaken nprefix-ntaken-ntaken-alt*)

lemma *in-guarantee-in-persistence*:
assumes *finitary S*
shows *σ* ∈ *Ep-i S* ⟷ *σ* ∈ *Pp-i* (*Ep-f S*)
using *assms*
unfolding *Ep-f-def Ep-i-def finitary-def Inf-def Ep-def Pp-i-def Pp-def Fin-def* **apply** *auto*
using *nprefix-eq-exists-ntaken* **apply** *blast*
by (*meson nprefix-eq-exists-ntaken nprefix-refl nprefix-trans*)

lemma *safety-eq-response*:
assumes *finitary S*
shows *Ap-i S = Rp-i (Ap-f S)*
using *assms in-safety-in-response*
by *auto*

lemma *guarantee-eq-response*:
assumes *finitary S*
shows *Ep-i S = Rp-i (Ep-f S)*
using *assms in-guarantee-in-response*
by *auto*

lemma *safety-eq-persistence*:
assumes *finitary S*
shows *Ap-i S = Pp-i (Ap-f S)*
using *assms in-safety-in-persistence*
by *auto*

lemma *guarantee-eq-persistence*:
assumes *finitary S*
shows *Ep-i S = Pp-i (Ep-f S)*
using *assms in-guarantee-in-persistence*
by *auto*

lemma *safety-is-response*:
assumes
safety-prop S
shows *response-prop S*
using *assms* **unfolding** *safety-prop-def response-prop-def*
by (*metis Ap-f-def Int-commute inf.cobounded1 safety-eq-response subseteq-Fin-eq-finitary*)

lemma *guarantee-is-response*:
assumes
guarantee-prop S
shows *response-prop S*
using *assms* **unfolding** *guarantee-prop-def response-prop-def*
by (*metis Ep-f-def Int-commute guarantee-eq-response inf.cobounded1 subseteq-Fin-eq-finitary*)

lemma *safety-is-persistence*:
assumes
safety-prop S
shows *persistence-prop S*
using *assms* **unfolding** *safety-prop-def persistence-prop-def*
by (*metis Ap-f-def Diff-Diff-Int finitary-Fin-complement inf.commute safety-eq-persistence*)

lemma *guarantee-is-persistence*:

assumes
guarantee-prop S
shows *persistence-prop S*
using *assms* **unfolding** *guarantee-prop-def persistence-prop-def*
by (*metis Diff-Diff-Int Ep-f-def Int-commute finitary-Fin-complement guarantee-eq-persistence*)

7.8.9 obligation

lemma *obligation-defs*:
assumes *length LS1 = length LS2*
shows *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*)))
proof −
have *1*: *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)!*s*)
by (*metis inter-list-defs*)
have *2*: *length* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) = *length LS1*
using *assms* **by** *simp*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)!*s*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*)))
using *assms* **by** *auto*
show *?thesis*
using *1 3* **by** *argo*
qed

lemma *obligation-defs-alt*:
assumes *length LS1 = length LS2*
shows *inter-list* (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ep-i* (*LS1*!*s*)) ∪ (*Ap-i* (*LS2*!*s*)))
proof −
have *1*: *inter-list* (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS1 LS2*)!*s*)
by (*metis inter-list-defs*)
have *2*: *length* (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS1 LS2*) = *length LS1*
using *assms* **by** *simp*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS1 LS2*)!*s*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ep-i* (*LS1*!*s*)) ∪ (*Ap-i* (*LS2*!*s*)))
using *assms* **by** *auto*
show *?thesis*
using *1 3* **by** *argo*
qed

lemma *obligation-defs-swap*:
assumes *length LS1 = length LS2*
shows *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) =
inter-list (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS2 LS1*)

proof −
have *1*: *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*)))
using *assms obligation-defs* **by** *blast*
have *2*: (⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*))) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ep-i* (*LS2*!*s*)) ∪ (*Ap-i* (*LS1*!*s*)))
by *blast*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ep-i* (*LS2*!*s*)) ∪ (*Ap-i* (*LS1*!*s*))) =
(⋂ *s* ∈ {*y* . *y* < *length LS2*}. (*Ep-i* (*LS2*!*s*)) ∪ (*Ap-i* (*LS1*!*s*)))

using *assms* **by** *presburger*
have *4*: (⋂ *s* ∈ {*y* . *y* < *length LS2*}. (*Ep-i* (*LS2*!*s*)) ∪ (*Ap-i* (*LS1*!*s*))) =
inter-list (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i S2*)) *LS2 LS1*)
using *assms obligation-defs-alt*[*of LS2 LS1*] **by** *argo*
show *?thesis*
using *1 2 3 4* **by** *presburger*
qed

lemma *complement-inter-list-1-Ap-Ep*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 >*0*
length LS1 = *length LS2*
shows *Inf* − *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) (*LS1*) (*LS2*)) =
union-list (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*))
proof −
have *2*: *length* (*map* ((−) *Inf*) (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) *LS1 LS2*)) =
length (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*))
by *auto*
have *3*: ⋀*j*. *j*< *length* (*map* ((−) *Inf*) (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) *LS1 LS2*)) ⟶
nth (*map* ((−) *Inf*) (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) *LS1 LS2*)) *j* =
nth (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*)) *j*
by *simp*
have *4*: (*map* ((−) *Inf*) (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) *LS1 LS2*)) =
(*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*))
using *2 3* **using** *nth-equalityI* **by** *blast*
show *?thesis*
by (*metis* (*no-types*, *lifting*) *4 assms*(*3*,*4*) *complement-inter-list length-map map-fst-zip*)
qed

lemma *complement-inter-list-2-Ap-Ep*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = *length LS2*
shows *union-list* (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*)) =
union-list (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∩ ((*Ap-i S2*))) (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*))

proof −
have *1*: *length* (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*)) =
length (*map2* (λ *S1 S2*. ((*Ep-i* (*Fin* − *S1*))) ∩ ((*Ap-i* (*Fin* − *S2*)))) (*LS1*) (*LS2*))
by *simp*
have *2*: ⋀*j*. *j* < *length* (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*)) ⟶
nth (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*)) *j* =
nth (*map2* (λ *S1 S2*. ((*Ep-i* (*Fin* − *S1*))) ∩ ((*Ap-i* (*Fin* − *S2*)))) (*LS1*) (*LS2*)) *j*
apply *simp*
by (*metis Ap-i-Fin Ap-i-inter-Inf Ap-inter Diff-Diff-Int Diff-Un Diff-subset Ep-i-eqv-Ap-i Ep-i-inter-Inf assms*(*2*) *finitary-inter*
finitary-list-defs inf-commute)
have *3*: (*map2* (λ *S1 S2*. ((*Ep-i* (*Fin* − *S1*))) ∩ ((*Ap-i* (*Fin* − *S2*)))) (*LS1*) (*LS2*)) =
(*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∩ ((*Ap-i S2*))) (*map* (λ*S*. *Fin*−*S*) *LS1*) (*map* (λ *S*. *Fin*−*S*) *LS2*))
by (*simp add*: *map-equality-iff*)
show *?thesis*
using *3 1 2 nth-equalityI* **by** *metis*
qed

lemma *obligation-defs-alt-add-Ep-i*:
assumes *length LS1* = *length LS2*
shows *inter-list* (*map2* (λ *S1 S2*. (*Ep-i* (*S1* ∪ *a*)) ∪ (*Ap-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ep-i* (*LS1*!*s* ∪ *a*)) ∪ (*Ap-i* (*LS2*!*s*)))
proof −
have *1*: *inter-list* (*map2* (λ *S1 S2*. (*Ep-i* (*S1* ∪ *a*)) ∪ (*Ap-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ep-i* (*S1* ∪ *a*)) ∪ (*Ap-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Ep-i* (*S1* ∪ *a*)) ∪ (*Ap-i S2*)) *LS1 LS2*)!*s*)
by (*metis inter-list-defs*)
have *2*: *length* (*map2* (λ *S1 S2*. (*Ep-i* (*S1* ∪ *a*)) ∪ (*Ap-i S2*)) *LS1 LS2*) = *length LS1*
using *assms* **by** *simp*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ep-i* (*S1* ∪ *a*)) ∪ (*Ap-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Ep-i* (*S1* ∪ *a*)) ∪ (*Ap-i S2*)) *LS1 LS2*)!*s*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ep-i* (*LS1*!*s* ∪ *a*)) ∪ (*Ap-i* (*LS2*!*s*)))
using *assms* **by** *auto*
show *?thesis*
using *1 3* **by** *argo*
qed

lemma *obligation-defs-alt-add-Ap-i*:
assumes *length LS1* = *length LS2*
shows *inter-list* (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ep-i* (*LS1*!*s*)) ∪ (*Ap-i* (*Ap-f* (*LS2*!*s*) ∪ *Ap-f b*)))
proof −
have *1*: *inter-list* (*map2* (λ *S1 S2*. (*Ep-i S1*) ∪ (*Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ep-i* (*S1*)) ∪ (*Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Ep-i* (*S1*)) ∪ (*Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*)!*s*)
by (*metis inter-list-defs*)
have *2*: *length* (*map2* (λ *S1 S2*. (*Ep-i* (*S1*)) ∪ (*Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*) = *length LS1*
using *assms* **by** *simp*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Ep-i* (*S1*)) ∪ (*Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*)}.

```
        (map2 (λ S1 S2. (Ep-i (S1)) ∪ (Ap-i (Ap-f S2 ∪ Ap-f b))) LS1 LS2)!s) =
       (⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (Ap-f (LS2!s) ∪ Ap-f b)))
  using assms by auto
 show ?thesis
 using 1 3 by argo
qed


lemma inter-list-add-Ep-i:
 assumes finitary-list LS1
         finitary-list LS2
         length LS1 = length LS2
         finitary a
 shows    (inter-list (map2 (λ S1 S2. ((Ep-i S1))  ∪ ( (Ap-i S2))) LS1 LS2) ∪ Ep-i a) =
          inter-list (map2 (λ S1 S2. ((Ep-i S1 )  ∪ ( (Ap-i S2)))) (map (λ S. S ∪ a) LS1)  LS2)
proof −
 have 1: (inter-list (map2 (λ S1 S2. ((Ep-i S1))  ∪ ( (Ap-i S2))) LS1 LS2) ∪ Ep-i a) =
         ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s))) ∪ Ep-i a)

   using assms obligation-defs-alt[of LS1 LS2 ]
     by argo
 have 2: ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s))) ∪ Ep-i a) =
         ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s)) ∪ (Ep-i a) ) )
  by blast
 have 3: ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s)) ∪ (Ep-i a) ) ) =
         ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s ∪ a)) ∪ (Ap-i (LS2!s))  ) )
  using Ep-union by blast
 have 4: ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s ∪ a)) ∪ (Ap-i (LS2!s))  ) ) =
          inter-list (map2 (λ S1 S2. ((Ep-i (S1 ∪ a) )  ∪ ( (Ap-i S2)))) LS1 LS2)
    using obligation-defs-alt-add-Ep-i assms by blast
 have 5: inter-list (map2 (λ S1 S2. ((Ep-i (S1 ∪ a) )  ∪ ( (Ap-i S2)))) LS1 LS2) =
          inter-list (map2 (λ S1 S2. ((Ep-i S1 )  ∪ ( (Ap-i S2)))) (map (λ S. S ∪ a) LS1)  LS2)
  by (simp add: map-zip-map)
 show ?thesis
 using 1 2 3 4 5 by argo
qed


lemma inter-list-add-Ap-i:
 assumes finitary-list LS1
         finitary-list LS2
         length LS1 = length LS2
         finitary b
 shows    (inter-list (map2 (λ S1 S2. ((Ep-i S1))  ∪ ( (Ap-i S2))) LS1 LS2) ∪ Ap-i b) =
          inter-list (map2 (λ S1 S2. ((Ep-i S1 )  ∪ ( (Ap-i S2)))) LS1  (map (λS. Ap-f S ∪ Ap-f b) LS2))
proof −
 have 1: (inter-list (map2 (λ S1 S2. ((Ep-i S1))  ∪ ( (Ap-i S2))) LS1 LS2) ∪ Ap-i b) =
         ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s))) ∪ Ap-i b)
  using assms obligation-defs-alt[of LS1 LS2 ] by argo
 have 2: ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s))) ∪ Ap-i b) =
```

((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s)) ∪ (Ap-i b)))
by *blast*
have *3*: ((⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (LS2!s)) ∪ (Ap-i b))) =
(⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (Ap-f (LS2!s) ∪ (Ap-f b))))
by (*simp add*: *Ap-i-union sup-assoc*)
have *4*: (⋂ s ∈ {y . y < length LS1}. (Ep-i (LS1!s)) ∪ (Ap-i (Ap-f (LS2!s) ∪ (Ap-f b)))) =
inter-list (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ *Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*)
by (*simp add*: *assms*(*3*) *obligation-defs-alt-add-Ap-i*)
have *5*: (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ *Ap-i* (*Ap-f S2* ∪ *Ap-f b*))) *LS1 LS2*) =
(*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ ((*Ap-i S2*)))) *LS1* (*map* (λ*S*. *Ap-f S* ∪ *Ap-f b*) *LS2*))
by (*simp add*: *map-equality-iff*)
show *?thesis*
using *1 2 3 4 5* **by** *argo*
qed

lemma *inter-list-add-Ep-i-Ap-i*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = *length LS2*
finitary a
finitary b
shows *inter-list* (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∪ ((*Ap-i S2*))) *LS1 LS2*) ∪ (*Ep-i a* ∩ *Ap-i b*) =
(*inter-list* (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ ((*Ap-i S2*)))) (*map* (λ *S*. *S* ∪ *a*) *LS1*) *LS2*) ∩
inter-list (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ ((*Ap-i S2*)))) *LS1* (*map* (λ*S*. *Ap-f S* ∪ *Ap-f b*) *LS2*)))
using *assms inter-list-add-Ep-i inter-list-add-Ap-i*
by *blast*

lemma *inter-list-add-Ep-i-Ap-i-cat*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = *length LS2*
finitary a
finitary b
shows (*inter-list* (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ ((*Ap-i S2*)))) (*map* (λ *S*. *S* ∪ *a*) *LS1*) *LS2*) ∩
inter-list (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ ((*Ap-i S2*)))) *LS1* (*map* (λ*S*. *Ap-f S* ∪ *Ap-f b*) *LS2*))) =
inter-list (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ ((*Ap-i S2*))))
((*map* (λ *S*. *S* ∪ *a*) *LS1*)@*LS1*) (*LS2*@(*map* (λ*S*. *Ap-f S* ∪ *Ap-f b*) *LS2*)))
using *assms* **by** (*simp add*: *inter-list-append*)

lemma *inter-list-add-Ep-i-Ap-i-add-left-right*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = *length LS2*
finitary a
finitary b
shows *inter-list* (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∪ ((*Ap-i S2*))) *LS1 LS2*) ∪ (*Ep-i a* ∩ *Ap-i b*) =
inter-list (*map2* (λ *S1 S2*. ((*Ep-i S1*) ∪ ((*Ap-i S2*)))) (*add-left a LS1*) (*add-right b LS2*))

using *assms*
by (*simp add*: *add-left-def add-right-def inter-list-add-Ep-i-Ap-i inter-list-add-Ep-i-Ap-i-cat*)

lemma *length-add-left*:
shows *length* (*add-left a LS1*) = *2*∗*length LS1*
unfolding *add-left-def* **by** *auto*

lemma *length-add-rigth*:
shows *length* (*add-right a LS1*) = *2*∗*length LS1*
unfolding *add-right-def* **by** *auto*

lemma *length-add-right-list-Nil*:
length (*add-right-list* [] *LS1*) = *length LS1*
by *simp*

lemma *length-add-right-list-append*:
shows *length* (*add-right-list LS2* (*LS1* @ *LS3*)) =
length (*add-right-list LS2 LS1*) +
length (*add-right-list LS2 LS3*)
proof (*induction LS2 arbitrary*: *LS1 LS3*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **by** (*simp add*: *add-right-def*)
qed

lemma *length-add-right-list-equal*:
assumes *length LS1* = *length LS3*
shows *length* (*add-right-list LS2 LS3*) = *length* (*add-right-list LS2 LS1*)
using *assms*
proof (*induction LS2 arbitrary*: *LS1 LS3*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **by** (*simp add*: *length-add-rigth*)
qed

lemma *length-add-right-list-Cons*:
length (*add-right-list* (*a* # *LS2*) *LS1*) = *2*∗(*length* (*add-right-list LS2 LS1*))
proof −
have *1*: *length* (*add-right-list* (*a* # *LS2*) *LS1*) = *length* (*add-right-list LS2* (*add-right a LS1*))
by *simp*
have *2*: *length* (*add-right-list LS2* (*add-right a LS1*)) =
length (*add-right-list LS2* (*LS1* @ *map* (λ*S*. *Ap-f S* ∪ *Ap-f a*) *LS1*))
unfolding *add-right-def* **by** *auto*
have *3*: *length* (*add-right-list LS2* (*LS1* @ *map* (λ*S*. *Ap-f S* ∪ *Ap-f a*) *LS1*)) =
length (*add-right-list LS2 LS1*) + *length* (*add-right-list LS2* (*map* (λ*S*. *Ap-f S* ∪ *Ap-f a*) *LS1*))

```
  by (simp add: length-add-right-list-append)
 have 4: length  (add-right-list LS2 (map (λS. Ap-f S ∪ Ap-f a) LS1)) =
          length  (add-right-list LS2 LS1)
  by (metis length-add-right-list-equal length-map)
 show ?thesis
 by (simp add: 2 3 4)
qed

lemma length-add-right-list:
 (length (add-right-list LS2 LS1)) = 2^(length LS2) * length LS1
proof (induction LS2 arbitrary: LS1)
case Nil
then show ?case by simp
next
case (Cons a LS2)
then show ?case
by (metis length-Cons length-add-right-list-Cons mult.assoc  power-Suc)
qed



lemma length-add-right-left-list:
 assumes length LS2 = length LS1
         length X = length Y
 shows length (add-right-list (map ((−) Fin) LS2) X) = length (add-left-list (map ((−) Fin) LS1) Y)
using assms
proof (induction LS2 LS1 arbitrary: X Y rule: list-induct2 )
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case
   proof −
    have 1: length (add-right-list (map ((−) Fin) (x # xs)) X) =
             length (add-right-list (map ((−) Fin) xs) (add-right (Fin − x) X))
      by simp
    have 2: length (add-left-list (map ((−) Fin) (y # ys)) Y) =
             length (add-left-list (map ((−) Fin) ys) (add-left (Fin − y) Y))
       by simp
    have 3: length (add-right-list (map ((−) Fin) xs) (add-right (Fin − x) X)) =
             length (add-left-list (map ((−) Fin) ys) (add-left (Fin − y) Y))
      using Cons.IH[of (add-right (Fin − x) X) (add-left (Fin − y) Y)]
       by (simp add: Cons.prems add-left-def add-right-def)
    show ?thesis
    by (simp add: 3)
  qed
qed


lemma finitary-list-map-union:
```

assumes *finitary-list LS1*
finitary x
shows *finitary-list* (*map* (λ*S*. *S* ∪ *x*) *LS1*)
using *assms*
proof (*induction LS1*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS1*)
then show *?case*
proof −
have *1*: *finitary a*
using *Cons.prems*(*1*) *finitary-list-defs* **by** *force*
have *2*: *finitary x*
by (*simp add*: *assms*(*2*))
have *3*: *finitary* (*a* ∪ *x*)
using *1 2 finitary-def* **by** *auto*
have *4*: *finitary-list* (*map* (λ*S*. *S* ∪ *x*) (*a* # *LS1*)) =
finitary (*a* ∪ *x*) ∧ *finitary-list* (*map* (λ*S*. *S* ∪ *x*) *LS1*)
unfolding *finitary-list-def* **apply** *simp*
by (*metis 2 Cons.IH Cons.prems*(*1*) *Cons-eq-rev-iff finitary-list-append finitary-list-def rev-append rev-rev-ident*)
show *?thesis*
using *3 4* **by** *auto*
qed
qed

lemma *finitary-list-map-inter*:
assumes *finitary-list LS1*
finitary x
shows *finitary-list* (*map* (λ*S*. *Ap-f S* ∪ *Ap-f x*) *LS1*)
using *assms*
proof (*induction LS1*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS1*)
then show *?case*
proof −
have *1*: *finitary a*
using *Cons.prems*(*1*) *finitary-list-defs* **by** *force*
have *2*: *finitary* (*Ap-f x*)
by (*simp add*: *Ap-f-def Fin-def finitary-def*)
have *25*: *finitary* (*Ap-f a*)
by (*simp add*: *Ap-f-def Fin-def finitary-def*)
have *3*: *finitary* ((*Ap-f a*) ∪ (*Ap-f x*))
using *1 2 finitary-def* **using** *25* **by** *auto*
have *4*: *finitary-list* (*map* (λ*S*. (*Ap-f S*) ∪ (*Ap-f x*)) (*a* # *LS1*)) =
finitary ((*Ap-f a*) ∪ (*Ap-f x*)) ∧ *finitary-list* (*map* (λ*S*. (*Ap-f S*) ∪ (*Ap-f x*)) *LS1*)

```
    unfolding finitary-list-def apply simp
    by (metis (no-types, lifting) ext Cons.IH Cons.prems(1) assms(2) finitary-list-def foldr-Cons o-apply)
  show ?thesis
   using 3 4 by auto
  qed
qed


lemma finitary-list-add-left:
 assumes finitary-list LS1
         finitary a
 shows finitary-list (add-left a LS1)
using assms unfolding add-left-def
 by (simp add: finitary-list-append finitary-list-map-union)


lemma finitary-list-add-right:
 assumes finitary-list LS2
         finitary b
 shows finitary-list (add-right b LS2)
using assms unfolding add-right-def
by (simp add: finitary-list-append finitary-list-map-inter)


lemma inter-list-union-list-absorb:
 assumes finitary-list LS1
         finitary-list LS2
         length LS1 = length LS2
         finitary-list LS3
         finitary-list LS4
         length LS3 = length LS4
 shows inter-list (map2 (λ S1 S2. ((Ep-i S1))  ∪ ( (Ap-i S2))) LS1 LS2) ∪
       union-list (map2 (λ S1 S2. ((Ep-i S1 )  ∩ ( (Ap-i S2)))) LS3 LS4) =
        inter-list (map2 (λ S1 S2. ((Ep-i S1))  ∪ ( (Ap-i S2))) (add-left-list LS3 LS1) (add-right-list LS4
LS2) )
using assms
proof (induction LS3 arbitrary: LS4 LS1 LS2)
case Nil
then show ?case unfolding union-list-def by simp
next
case (Cons a LS3a)
then show ?case
 proof −
  have 1: ∃ b LS4b. LS4 = b # LS4b
    by (metis Cons.prems(6) length-Suc-conv)
  obtain b LS4b where 2: LS4 = b # LS4b
    using 1 by blast
  have 3: union-list (map2 (λ S1 S2. ((Ep-i S1 )  ∩ ( (Ap-i S2)))) (a#LS3a) LS4) =
          ((Ep-i a )  ∩ ( (Ap-i b))) ∪ union-list (map2 (λ S1 S2. ((Ep-i S1 )  ∩ ( (Ap-i S2)))) LS3a LS4b)
    using 2 unfolding union-list-def by simp
```

have *4*: *finitary a*
using *Cons.prems*(*4*) *finitary-list-defs* **by** *force*
have *5*: *finitary b*
using *2 Cons.prems*(*5*) *finitary-list-defs* **by** *fastforce*
have *6*: *inter-list* (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∪ ((*Ap-i S2*))) *LS1 LS2*) ∪ ((*Ep-i a*) ∩ ((*Ap-i b*))) =
inter-list (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∪ ((*Ap-i S2*))) (*add-left a LS1*) (*add-right b LS2*))
using *inter-list-add-Ep-i-Ap-i-add-left-right*[*of LS1 LS2 a b*]
using *4 5 Cons.prems*(*1*,*2*,*3*) **by** *linarith*
have *7*: *inter-list* (*map2* (λ*x y*. *Ep-i x* ∪ *Ap-i y*) (*add-left-list* (*a* # *LS3a*) *LS1*) (*add-right-list* (*b* # *LS4*) *LS2*)) =
inter-list (*map2* (λ*x y*. *Ep-i x* ∪ *Ap-i y*) ((*add-left-list LS3a* (*add-left a LS1*))) ((*add-right-list LS4* (*add-right b LS2*))))
by *simp*
have *8*: *finitary-list LS3a*
by (*metis Cons.prems*(*4*) *Cons-eq-append-conv finitary-list-append self-append-conv2*)
have *9*: *finitary-list LS4b*
by (*metis 2 Cons.prems*(*5*) *Suc-less-eq2 finitary-list-defs length-Cons nth-Cons-Suc*)
have *10*: *finitary-list* (*add-left a LS1*)
unfolding *add-left-def finitary-list-def* **using** *Cons.prems*
by (*metis 4 add-left-def finitary-list-add-left finitary-list-def*)
have *11*: *finitary-list* (*add-right b LS2*)
by (*simp add*: *5 Cons.prems*(*2*) *finitary-list-add-right*)
have *12*: *inter-list* (*map2* (λ*x y*. *Ep-i x* ∪ *Ap-i y*) (*add-left a LS1*) (*add-right b LS2*)) ∪ *union-list* (*map2* (λ*x y*. *Ep-i x* ∩ *Ap-i y*) *LS3a LS4b*) =
inter-list (*map2* (λ*x y*. *Ep-i x* ∪ *Ap-i y*) (*add-left-list LS3a* (*add-left a LS1*)) (*add-right-list LS4b* (*add-right b LS2*)))
using *Cons.IH*[*of add-left a LS1 add-right b LS2 LS4b*]
using *2 8 9 Cons.prems*(*1*,*2*,*3*,*6*)
by (*metis 10 11 add-left-def add-right-def length-Cons length-append length-map nat.simps*(*1*))
show *?thesis*
using *12 2 3 6* **by** *auto*
qed
qed

lemma *finitary-list-add-right-list*:
assumes *finitary-list LS2*
finitary-list X
shows *finitary-list* (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) *X*)
using *assms*
proof (*induction LS2 arbitrary*: *X*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **apply** *simp*
by (*metis DiffD1 append-Cons append-Nil finitary-Fin finitary-def finitary-list-add-right finitary-list-append*)
qed

lemma *finitary-list-add-left-list*:
assumes *finitary-list LS1*
finitary-list X
shows *finitary-list* (*add-left-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) *X*)
using *assms*
proof (*induction LS1 arbitrary*: *X*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS1*)
then show *?case* **apply** *simp*
by (*metis Diff-iff Fin-def add-left-def append-Cons append-Nil finitary-def finitary-list-append finitary-list-map-union mem-Collect-eq*)
qed

lemma *length-add-right-list-gr-zer0*:
assumes *length LS2* > *0*
length X > *0*
shows *length* (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) *X*) >*0*
using *assms*
proof (*induction LS2 arbitrary*: *X*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **apply** *simp*
by (*metis add-right-def add-right-list.simps*(*1*) *append-is-Nil-conv list.map-disc-iff*)
qed

lemma *obligation-prop-subset-eq-Inf*:
assumes *obligation-prop S k*
shows *S* ⊆ *Inf*
proof −
obtain *LS1 LS2* **where** *1*: *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *length LS1* >*0* ∧ *length LS1* = *k* ∧
S = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)
using *assms obligation-prop-def* **by** *blast*
have *2*: *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) =
((⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*))))
by (*simp add*: *1 obligation-defs*)
have *3*: ⋀*s*. *s* ∈ {*y* . *y* < *length LS1*} ⟹ (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*)) ⊆ *Inf*
using *Ap-i-inter-Inf Ep-i-inter-Inf* **by** *blast*
have *4*: ((⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*)))) ⊆ *Inf*
using *1 3* **by** *blast*
show *?thesis* **using** *1 2 4* **by** *presburger*
qed

lemma *complement-obligation-prop*:
assumes *obligation-prop S k*

shows *obligation-prop* (*Inf* −*S*) (*2^k*)
proof −
obtain *LS1 LS2* **where** *1*: *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *length LS1* >*0* ∧ *length LS1* = *k* ∧
S = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)
using *assms obligation-prop-def* **by** *blast*
have *2*: *Inf* − *S* = *Inf* − (*inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*))
by (*simp add*: *1*)
have *3*: *Inf* − (*inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)) =
union-list (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*))
using *1 complement-inter-list-1-Ap-Ep* **by** *blast*
have *4*: *union-list* (*map2* (λ *S1 S2*. *Inf* − ((*Ap-i S1*) ∪ (*Ep-i S2*))) (*LS1*) (*LS2*)) =
union-list (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∩ ((*Ap-i S2*))) (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*))
by (*simp add*: *1 complement-inter-list-2-Ap-Ep*)
have *5*: *Fs* = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) [*Fs*] [*Fs*])
unfolding *inter-list-def* **by** (*simp add*: *Ap-i-Fs Ep-i-Fs*)
have *6*: *union-list* (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∩ ((*Ap-i S2*))) (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*)) =
inter-list (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) [*Fs*] [*Fs*]) ∪
union-list (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∩ ((*Ap-i S2*))) (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*))
using *5 Fs-def* **by** *auto*
have *7*: *finitary-list* [*Fs*]
by (*simp add*: *finitary-Fs finitary-list-defs*)
have *8*: *finitary-list* (*map* ((−) *Fin*) *LS1*)
by (*simp add*: *Fin-def finitary-def finitary-list-defs*)
have *9*: *finitary-list* (*map* ((−) *Fin*) *LS2*)
by (*simp add*: *Fin-def finitary-def finitary-list-defs*)
have *10*: *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) [*Fs*] [*Fs*]) ∪
union-list (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∩ ((*Ap-i S2*))) (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*)) =
inter-list (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∪ ((*Ap-i S2*)))
(*add-left-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) [*Fs*]) (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*]))
using *inter-list-union-list-absorb*[*of* [*Fs*] [*Fs*] (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*)]
by (*metis 1 7 8 9 length-map obligation-defs-swap*)
have *11*: *length* (*add-right-list* (*map* ((−) *Fin*) *LS2*) [*Fs*]) = *length* (*add-left-list* (*map* ((−) *Fin*) *LS1*) [*Fs*])
by (*simp add*: *1 length-add-right-left-list*)
have *12*: *inter-list* (*map2* (λ *S1 S2*. ((*Ep-i S1*)) ∪ ((*Ap-i S2*)))
(*add-left-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) [*Fs*]) (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*])) =
inter-list (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*))
(*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*]) (*add-left-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) [*Fs*]))
using *obligation-defs-swap*[*of* (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*])
(*add-left-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) [*Fs*])]
using *11* **by** *blast*
have *13*: *finitary-list* (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*])
by (*simp add*: *1 7 finitary-list-add-right-list*)
have *14*: *finitary-list* (*add-left-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) [*Fs*])
by (*simp add*: *1 7 finitary-list-add-left-list*)
have *15*: *length* (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*]) >*0*

by (*metis 1 length-Cons length-add-right-list-gr-zer0*
zero-less-Suc)
have *16*: *length* (*add-right-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*]) = *2*^*k*
by (*metis* (*no-types*, *lifting*) *1 length-Cons length-add-right-list length-map list.size*(*3*) *mult-numeral-1-right numeral-1-eq-Suc-0*)
show *?thesis* **unfolding** *obligation-prop-def*
by (*metis 1 10 11 12 13 14 15 16 3 4 6 Diff-subset subseteq-Inf-eq-infinitary*)
qed

lemma *complement-obligation-prop-a*:
assumes *S*⊆*Inf*
obligation-prop (*Inf* − *S*) *k*
shows *obligation-prop S* (*2*^*k*)
using *assms*
by (*metis Diff-Diff-Int complement-obligation-prop inf.absorb-iff2*)

lemma *obligation-prop-inter*:
assumes *obligation-prop S1 k*
obligation-prop S2 l
shows *obligation-prop* (*S1* ∩ *S2*) (*k*+*l*)
proof −
have *1*: (∃ *LS1 LS2*. *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *0* < *length LS1* ∧ *length LS1* = *k* ∧
S1 = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*))
using *assms obligation-prop-def*[*of S1*] **by** *blast*
obtain *LS1 LS2* **where** *2*: *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *0* < *length LS1* ∧ *length LS1* = *k* ∧
S1 = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*)
using *1* **by** *blast*
have *3*: (∃ *LS1 LS2*. *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *0* < *length LS1* ∧ *length LS1*=*l* ∧
S2 = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*))
using *assms obligation-prop-def* **by** *blast*
obtain *LS3 LS4* **where** *4*: *finitary-list LS3* ∧ *finitary-list LS4* ∧
length LS3 = *length LS4* ∧ *0* < *length LS3* ∧ *length LS3* = *l* ∧
S2 = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS3 LS4*)
using *3* **by** *blast*
have *5*: *S1* ∩ *S2* = *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) ∩
inter-list (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS3 LS4*)
using *2 4* **by** *blast*
have *6*: *inter-list* (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS1 LS2*) ∩
inter-list (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) *LS3 LS4*) =
inter-list (*map2* (λ *S1 S2*. (*Ap-i S1*) ∪ (*Ep-i S2*)) (*LS1*@*LS3*) (*LS2*@*LS4*))
by (*simp add*: *2 inter-list-append*)
have *7*: *finitary-list* (*LS1*@*LS3*)
by (*simp add*: *2 4 finitary-list-append*)
have *8*: *finitary-list* (*LS2*@*LS4*)
by (*simp add*: *2 4 finitary-list-append*)

have *9*: *length* (*LS1*@*LS3*) = *length* (*LS2*@*LS4*)
using *2 4* **by** *auto*
have *10*: *0* < *length* (*LS1*@*LS3*)
using *2 4* **by** *blast*
have *11*: *length* (*LS1*@*LS3*) = *k*+*l*
using *2 4* **by** *fastforce*
show *?thesis* **unfolding** *obligation-prop-def*
using *2 4 6 7 8 9 11*
by (*metis 10 assms*(*1*) *le-infI1 obligation-prop-subset-eq-Inf subseteq-Inf-eq-infinitary*)
qed

lemma *obligation-prop-union*:
assumes *obligation-prop S1 k*
obligation-prop S2 l
shows *obligation-prop* (*S1* ∪ *S2*) (*2*^(*2*^*k*) + (*2*^*l*)))
proof −
have *1*: *obligation-prop* (*Inf* −*S1*) (*2*^*k*)
by (*simp add*: *assms*(*1*) *complement-obligation-prop*)
have *2*: *obligation-prop* (*Inf* −*S2*) (*2*^*l*)
by (*simp add*: *assms*(*2*) *complement-obligation-prop*)
have *3*: *obligation-prop* ((*Inf* −*S1*) ∩ (*Inf* −*S2*)) ((*2*^*k*) + (*2*^*l*))
by (*simp add*: *1 2 obligation-prop-inter*)
have *4*: *obligation-prop* (*Inf* −((*Inf* −*S1*) ∩ (*Inf* −*S2*))) (*2*^(*2*^*k*) + (*2*^*l*)))
by (*simp add*: *3 complement-obligation-prop*)
have *5*: (*Inf* −((*Inf* −*S1*) ∩ (*Inf* −*S2*))) = (*S1* ∪ *S2*)
by (*metis Diff-Diff-Int Diff-Int Diff-partition Un-Int-eq*(*1*) *assms*(*1,2*) *obligation-prop-subset-eq-Inf*)
show *?thesis*
using *4 5* **by** *auto*
qed

lemma *safety-is-obligation*:
assumes *safety-prop S*
shows *obligation-prop S 1*
proof −
obtain *S1* **where** *1*: *finitary S1* ∧ *infinitary S* ∧ *S* = *Ap-i S1*
using *assms safety-prop-def* **by** *blast*
have *2*: *S* = *Ap-i S1* ∪ *Ep-i Fs*
using *1 Ep-i-Fs Fs-def* **by** *fastforce*
have *3*: *finitary-list* [*S1*]
by (*simp add*: *1 finitary-list-defs*)
have *4*: *finitary-list* [*Fs*]
by (*simp add*: *finitary-Fs finitary-list-defs*)
have *5*: *length* [*S1*] = *length* [*Fs*]
by *simp*
have *6*: *0* < *length* [*S1*]
by *simp*
have *7*: *length* [*S1*] = *1*
by *simp*
have *8*: *S* = *inter-list* (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) [*S1*] [*Fs*])

unfolding *inter-list-def* **using** *2* **by** *auto*
show *?thesis* **unfolding** *obligation-prop-def*
using *1 3 4 5 7 8* **by** *blast*
qed

lemma *guarantee-is-obligation*:
assumes *guarantee-prop S*
shows *obligation-prop S 1*
proof −
obtain *S1* **where** *1*: *finitary S1* ∧ *infinitary S* ∧ *S* = *Ep-i S1*
using *assms guarantee-prop-def* **by** *blast*
have *2*: *S* = *Ap-i Fs* ∪ *Ep-i S1*
using *1 Ap-i-Fs Fs-def* **by** *fastforce*
have *3*: *finitary-list* [*S1*]
by (*simp add*: *1 finitary-list-defs*)
have *4*: *finitary-list* [*Fs*]
by (*simp add*: *finitary-Fs finitary-list-defs*)
have *5*: *length* [*Fs*] = *length* [*S1*]
by *simp*
have *6*: *0* < *length* [*Fs*]
by *simp*
have *7*: *length* [*Fs*] = *1*
by *simp*
have *8*: *S* = *inter-list* (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) [*Fs*] [*S1*])
unfolding *inter-list-def* **using** *2* **by** *auto*
show *?thesis* **unfolding** *obligation-prop-def*
using *1 3 4 5 7 8* **by** *blast*
qed

lemma *obbligation-1-is-response*:
assumes *obligation-prop S 1*
shows *response-prop S*
proof −
obtain *LS1 LS2* **where** *1*: *finitary-list LS1* ∧
finitary-list LS2 ∧
length LS1 = *length LS2* ∧
0 < *length LS1* ∧ *length LS1* = *1* ∧ *infinitary S* ∧ *S* = *inter-list* (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) *LS1 LS2*)
using *assms* **unfolding** *obligation-prop-def* **by** *blast*
have *2*: ∃ *S1*. *LS1* = [*S1*]
by (*metis 1 One-nat-def length-0-conv length-Suc-conv*)
obtain *S1* **where** *3*: *LS1* = [*S1*]
using *2* **by** *blast*
have *4*: ∃ *S2*. *LS2* = [*S2*]
by (*metis 1 One-nat-def length-0-conv length-Suc-conv*)
obtain *S2* **where** *5*: *LS2* = [*S2*]
using *4* **by** *blast*
have *6*: *S* = *Ap-i S1* ∪ *Ep-i S2*
using *1 3 5* **unfolding** *inter-list-def* **by** *simp*
show *?thesis* **unfolding** *response-prop-def* **using** *6*

by (*metis 1 3 5 Ep-f-def finitary-list-defs*
guarantee-eq-response inf-le2 le-sup-iff nth-Cons-0
response-prop-def response-prop-union safety-is-response
safety-prop-def subseteq-Fin-eq-finitary
subseteq-Inf-eq-infinitary)
qed

lemma *obligation-k-is-response*:
assumes *obligation-prop S k*
shows *response-prop S*
using *assms*
proof (*induction k arbitrary*: *S*)
case *0*
then show *?case*
using *obligation-prop-def* **by** *blast*
next
case (*Suc k*)
then show *?case*
proof (*cases k=0*)
case *True*
then show *?thesis*
using *Suc.prems obbligation-1-is-response* **by** *auto*
next
case *False*
then show *?thesis*
proof −
obtain *LS1 LS2* **where** *1*: *finitary-list LS1* ∧
finitary-list LS2 ∧
length LS1 = length LS2 ∧
0 < length LS1 ∧ *length LS1* = (*Suc k*) ∧ *infinitary S* ∧ *S = inter-list* (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) *LS1 LS2*)
using *assms Suc.prems obligation-prop-def* **by** *blast*
have *2*: ∃ *LS11 a*. *LS1 = a # LS11*
using *1* **by** (*meson length-Suc-conv*)
obtain *a LS11* **where** *3*: *LS1 = a # LS11*
using *2* **by** *blast*
have *4*: ∃ *LS21 b*. *LS2 = b # LS21*
using *1* **by** (*metis length-Suc-conv*)
obtain *b LS21* **where** *5*: *LS2 = b # LS21*
using *4* **by** *blast*
have *51*: *length LS11 = length LS21*
using *1 3 5* **by** *auto*
have *6*: *S* = (*Ap-i a* ∪ (*Ep-i b*)) ∩ *inter-list* (*map2* (λ*x y*. *Ap-i x* ∪ *Ep-i y*) *LS11 LS21*)
using *3 5 1* **unfolding** *inter-list-def* **by** *simp*
have *7*: *response-prop* (*Ap-i a* ∪ (*Ep-i b*))
by (*metis 1 5 Ap-i-Fin Ap-i-inter-Inf Ap-inter Ep-i-inter-Inf finitary-list-defs guarantee-is-response*
guarantee-prop-def inf-le2 nth-Cons-0 response-prop-union safety-is-response safety-prop-def
subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary)
have *80*: ⋀*s*. *s* ∈ {*y* . *y < length LS1*} ⟹ (*Ap-i* (*LS1*!*s*)) ∪ (*Ep-i* (*LS2*!*s*)) ⊆ *Inf*
using *Ap-i-inter-Inf Ep-i-inter-Inf* **by** *blast*

```
    have 81: infinitary (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS1 LS2))
      using 1 by auto
    have 82: ⋀s. s ∈ {y . y < length LS11} ⟹ (Ap-i (LS11!s)) ∪ (Ep-i (LS21!s)) ⊆ Inf
      using Ap-i-inter-Inf Ep-i-inter-Inf by blast
    have 83: inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21) =
            (⋂ s∈{y. y < length LS11}. Ap-i (LS11 ! s) ∪ Ep-i (LS21 ! s))
      by (simp add: 51 obligation-defs)
    have 84: (⋂ s∈{y. y < length LS11}. Ap-i (LS11 ! s) ∪ Ep-i (LS21 ! s)) ⊆ Inf
       using 82
       by (metis (no-types, lifting) 1 5 51  False INF-lower2
        length-Cons length-greater-0-conv list.size(3) mem-Collect-eq nat.inject )
    have 8: infinitary (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21))
     using 83 84 subseteq-Inf-eq-infinitary by auto
    have 9: obligation-prop (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21)) k
     using 1 3 5  unfolding obligation-prop-def apply auto
     by (metis 8 False append-Nil append-eq-Cons-conv finitary-list-append length-0-conv)
    have 10: response-prop (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21))
      using Suc.IH[of (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21))]
      by (simp add: 9)
    show ?thesis
    by (simp add: 6 7 10 response-prop-inter)
   qed
  qed
qed

lemma obbligation-1-is-persistence:
 assumes obligation-prop S 1
 shows   persistence-prop S
proof −
 obtain LS1 LS2 where 1: finitary-list LS1 ∧
     finitary-list LS2 ∧
     length LS1 = length LS2 ∧
     0 < length LS1 ∧ length LS1 = 1 ∧ infinitary S ∧ S = inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS1
LS2)
    using assms unfolding obligation-prop-def by blast
 have 2: ∃ S1. LS1 = [S1]
     by (metis 1 One-nat-def length-0-conv length-Suc-conv)
 obtain S1 where 3: LS1 = [S1]
   using 2 by blast
 have 4: ∃ S2. LS2 = [S2]
     by (metis 1 One-nat-def length-0-conv length-Suc-conv)
 obtain S2 where 5: LS2 = [S2]
   using 4 by blast
 have 6: S = Ap-i S1 ∪ Ep-i S2
   using 1 3 5 unfolding  inter-list-def by simp
 show ?thesis unfolding persistence-prop-def using 6
  by (metis Ap-f-def Ap-i-inter-Inf Ap-i-union Ep-f-def Ep-i-inter Ep-i-inter-Inf guarantee-is-persistence
   guarantee-prop-def inf.idem inf-commute inf-le1 persistence-prop-def persistence-prop-union safety-is-persistence
    safety-prop-def subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary sup-idem)
qed
```

lemma *obligation-k-is-persistence*:
assumes *obligation-prop S k*
shows *persistence-prop S*
using *assms*
proof (*induction k arbitrary*: *S*)
case *0*
then show *?case*
using *obligation-prop-def* **by** *blast*
next
case (*Suc k*)
then show *?case*
proof (*cases k=0*)
case *True*
then show *?thesis*
using *Suc.prems obbligation-1-is-persistence* **by** *auto*
next
case *False*
then show *?thesis*
proof −
obtain *LS1 LS2* **where** *1*: *finitary-list LS1* ∧
finitary-list LS2 ∧
length LS1 = length LS2 ∧
0 < length LS1 ∧ *length LS1 = (Suc k)* ∧ *infinitary S* ∧ *S = inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS1 LS2*)
using *assms Suc.prems obligation-prop-def* **by** *blast*
have *2*: ∃ *LS11 a. LS1 = a # LS11*
using *1* **by** (*meson length-Suc-conv*)
obtain *a LS11* **where** *3*: *LS1 = a # LS11*
using *2* **by** *blast*
have *4*: ∃ *LS21 b. LS2 = b # LS21*
using *1* **by** (*metis length-Suc-conv*)
obtain *b LS21* **where** *5*: *LS2 = b # LS21*
using *4* **by** *blast*
have *51*: *length LS11 = length LS21*
using *1 3 5* **by** *auto*
have *6*: *S = (Ap-i a ∪ (Ep-i b)) ∩ inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21)*
using *3 5 1* **unfolding** *inter-list-def* **by** *simp*
have *7*: *persistence-prop (Ap-i a ∪ (Ep-i b))*
by (*metis 1 5 Ap-i-Fin Ap-i-def Ap-inter Ep-i-def finitary-list-defs guarantee-is-persistence guarantee-prop-def*
inf.absorb-iff2 inf-commute inf-le2 nth-Cons-0 persistence-prop-union safety-is-persistence safety-prop-def
subseteq-Fin-eq-finitary subseteq-Inf-eq-infinitary)
have *80*: ⋀*s. s* ∈ {*y . y < length LS1*} ⟹ (*Ap-i (LS1!s)*) ∪ (*Ep-i (LS2!s)*) ⊆ *Inf*
using *Ap-i-inter-Inf Ep-i-inter-Inf* **by** *blast*
have *81*: *infinitary (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS1 LS2))*
using *1* **by** *auto*
have *82*: ⋀*s. s* ∈ {*y . y < length LS11*} ⟹ (*Ap-i (LS11!s)*) ∪ (*Ep-i (LS21!s)*) ⊆ *Inf*
using *Ap-i-inter-Inf Ep-i-inter-Inf* **by** *blast*

```
    have 83: inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21) =
              (⋂s∈{y. y < length LS11}. Ap-i (LS11 ! s) ∪ Ep-i (LS21 ! s))
      by (simp add: 51 obligation-defs)
    have 84: (⋂s∈{y. y < length LS11}. Ap-i (LS11 ! s) ∪ Ep-i (LS21 ! s)) ⊆ Inf
        using 82
        by (metis (no-types, lifting) 1 5 51  False INF-lower2
          length-Cons length-greater-0-conv list.size(3) mem-Collect-eq nat.inject )
    have 8: infinitary (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21))
      using 83 84 subseteq-Inf-eq-infinitary by auto
    have 9: obligation-prop (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21)) k
      using 1 3 5  unfolding obligation-prop-def apply auto
      by (metis 8 False append-Nil append-eq-Cons-conv finitary-list-append length-0-conv)
    have 10: persistence-prop (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21))
      using Suc.IH[of (inter-list (map2 (λx y. Ap-i x ∪ Ep-i y) LS11 LS21))]
      by (simp add: 9)
    show ?thesis
    by (simp add: 6 7 10 persistence-prop-inter)
   qed
  qed
qed
```

7.8.10 reactivity

```
lemma reactivity-defs:
assumes  length LS1 = length LS2
shows  inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2) =
      (⋂ s ∈ {y . y < length LS1}. (Rp-i (LS1!s)) ∪ (Pp-i (LS2!s)))
proof −
 have 1: inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2) =
         (⋂ s ∈ {y . y < length (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2)}.
           (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2)!s)
   by (metis inter-list-defs)
 have 2: length (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2) = length LS1
   using assms by simp
 have 3: (⋂ s ∈ {y . y < length (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2)}.
           (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2)!s) =
          (⋂ s ∈ {y . y < length LS1}. (Rp-i (LS1!s)) ∪ (Pp-i (LS2!s)))
   using assms by auto
 show ?thesis
 using 1 3 by argo
qed

lemma reactivity-defs-alt:
assumes  length LS1 = length LS2
shows  inter-list (map2 (λ S1 S2. (Pp-i S1) ∪ (Rp-i S2)) LS1 LS2) =
      (⋂ s ∈ {y . y < length LS1}. (Pp-i (LS1!s)) ∪ (Rp-i (LS2!s)))
proof −
 have 1: inter-list (map2 (λ S1 S2. (Pp-i S1) ∪ (Rp-i S2)) LS1 LS2) =
         (⋂ s ∈ {y . y < length (map2 (λ S1 S2. (Pp-i S1) ∪ (Rp-i S2)) LS1 LS2)}.
           (map2 (λ S1 S2. (Pp-i S1) ∪ (Rp-i S2)) LS1 LS2)!s)
```

by (*metis inter-list-defs*)
have *2*: *length* (*map2* (λ *S1 S2*. (*Pp-i S1*) ∪ (*Rp-i S2*)) *LS1 LS2*) = *length LS1*
using *assms* **by** *simp*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Pp-i S1*) ∪ (*Rp-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Pp-i S1*) ∪ (*Rp-i S2*)) *LS1 LS2*)!*s*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Pp-i* (*LS1*!*s*)) ∪ (*Rp-i* (*LS2*!*s*)))
using *assms* **by** *auto*
show *?thesis*
using *1 3* **by** *argo*
qed

lemma *reactivity-defs-swap*:
assumes *length LS1* = *length LS2*
shows *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS1 LS2*) =
inter-list (*map2* (λ *S1 S2*. (*Pp-i S1*) ∪ (*Rp-i S2*)) *LS2 LS1*)

proof −
have *1*: *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Rp-i* (*LS1*!*s*)) ∪ (*Pp-i* (*LS2*!*s*)))
using *assms reactivity-defs* **by** *blast*
have *2*: (⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Rp-i* (*LS1*!*s*)) ∪ (*Pp-i* (*LS2*!*s*))) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Pp-i* (*LS2*!*s*)) ∪ (*Rp-i* (*LS1*!*s*)))
by *blast*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Pp-i* (*LS2*!*s*)) ∪ (*Rp-i* (*LS1*!*s*))) =
(⋂ *s* ∈ {*y* . *y* < *length LS2*}. (*Pp-i* (*LS2*!*s*)) ∪ (*Rp-i* (*LS1*!*s*)))

using *assms* **by** *presburger*
have *4*: (⋂ *s* ∈ {*y* . *y* < *length LS2*}. (*Pp-i* (*LS2*!*s*)) ∪ (*Rp-i* (*LS1*!*s*))) =
inter-list (*map2* (λ *S1 S2*. (*Pp-i S1*) ∪ (*Rp-i S2*)) *LS2 LS1*)
using *assms reactivity-defs-alt*[*of LS2 LS1*] **by** *argo*
show *?thesis*
using *1 2 3 4* **by** *presburger*
qed

lemma *complement-inter-list-1-Rp-Pp*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 > *0*
length LS1 = *length LS2*
shows *Inf* − *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) (*LS1*) (*LS2*)) =
union-list (*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*))
proof −
have *2*: *length* (*map* ((−) *Inf*) (*map2* (λ*x y*. *Rp-i x* ∪ *Pp-i y*) *LS1 LS2*)) =
length (*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*))
by *auto*
have *3*: ⋀*j*. *j*< *length* (*map* ((−) *Inf*) (*map2* (λ*x y*. *Rp-i x* ∪ *Pp-i y*) *LS1 LS2*)) ⟶
nth (*map* ((−) *Inf*) (*map2* (λ*x y*. *Rp-i x* ∪ *Pp-i y*) *LS1 LS2*)) *j* =
nth (*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*)) *j*
by *simp*

have *4*: (*map* ((−) *Inf*) (*map2* (λ*x y*. *Rp-i x* ∪ *Pp-i y*) *LS1 LS2*)) =
(*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*))
using *2 3* **using** *nth-equalityI* **by** *blast*
show *?thesis*
by (*metis* (*lifting*) *4 assms*(*3*,*4*) *complement-inter-list length-greater-0-conv list.map-disc-iff map-snd-zip*)
qed

lemma *complement-inter-list-2-Rp-Pp*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = *length LS2*
shows *union-list* (*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*)) =
union-list (*map2* (λ *S1 S2*. ((*Pp-i S1*)) ∩ ((*Rp-i S2*))) (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*))
proof −
have *1*: *length* (*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*)) =
length (*map2* (λ *S1 S2*. ((*Pp-i* (*Fin* − *S1*))) ∩ ((*Rp-i* (*Fin* − *S2*)))) (*LS1*) (*LS2*))
by *simp*
have *2*: ⋀*j*. *j* < *length* (*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*)) ⟶
nth (*map2* (λ *S1 S2*. *Inf* − ((*Rp-i S1*) ∪ (*Pp-i S2*))) (*LS1*) (*LS2*)) *j* =
nth (*map2* (λ *S1 S2*. ((*Pp-i* (*Fin* − *S1*))) ∩ ((*Rp-i* (*Fin* − *S2*)))) (*LS1*) (*LS2*)) *j*
apply *simp*
by (*metis Diff-Un Pp-i-eqv-Rp-i Rp-i-eqv-Pp-i assms*(*1*,*2*) *finitary-list-defs subseteq-Fin-eq-finitary*)
have *3*: (*map2* (λ *S1 S2*. ((*Pp-i* (*Fin* − *S1*))) ∩ ((*Rp-i* (*Fin* − *S2*)))) (*LS1*) (*LS2*)) =
(*map2* (λ *S1 S2*. ((*Pp-i S1*)) ∩ ((*Rp-i S2*))) (*map* (λ*S*. *Fin*−*S*) *LS1*) (*map* (λ *S*. *Fin*−*S*) *LS2*))
by (*simp add*: *map-equality-iff*)
show *?thesis*
using *3 1 2 nth-equalityI* **by** *metis*
qed

lemma *reactivity-defs-alt-add-Pp-i*:
assumes *length LS1* = *length LS2*
shows *inter-list* (*map2* (λ *S1 S2*. (*Pp-i* (*S1* ∪ (*Fin* − (((*S1* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length LS1*}. (*Pp-i* (*LS1*!*s* ∪ (*Fin* − (((*LS1*!*s* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i* (*LS2*!*s*)))
proof −
have *1*: *inter-list* (*map2* (λ *S1 S2*. (*Pp-i* (*S1* ∪ (*Fin* − (((*S1* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i S2*)) *LS1 LS2*) =
(⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Pp-i* (*S1* ∪ (*Fin* − (((*S1* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Pp-i* (*S1* ∪ (*Fin* − (((*S1* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i S2*)) *LS1 LS2*)!*s*)
by (*metis inter-list-defs*)
have *2*: *length* (*map2* (λ *S1 S2*. (*Pp-i* (*S1* ∪ (*Fin* − (((*S1* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i S2*)) *LS1 LS2*) = *length LS1*
using *assms* **by** *simp*
have *3*: (⋂ *s* ∈ {*y* . *y* < *length* (*map2* (λ *S1 S2*. (*Pp-i* (*S1* ∪ (*Fin* − (((*S1* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i S2*)) *LS1 LS2*)}.
(*map2* (λ *S1 S2*. (*Pp-i* (*S1* ∪ (*Fin* − (((*S1* 𝒮 (*Fin* − *a*)) · *Empty*))))) ∪ (*Rp-i S2*)) *LS1 LS2*)!*s*) =

$(\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s \cup (Fin - (((LS1!s\ \mathcal{S}\ (Fin - a)) \cdot Empty))))) \cup (Rp\text{-}i\ (LS2!s)))$
using *assms* **by** *auto*
show *?thesis*
using *1 3* **by** *argo*
qed

lemma *reactivity-defs-alt-add-Rp-i*:
assumes *length LS1 = length LS2*
shows *inter-list (map2 (λ S1 S2. (Pp-i S1) ∪ (Rp-i (S2 ∪ b))) LS1 LS2) =*
$(\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (\ (LS2!s) \cup\ b)))$
proof −
have *1*: *inter-list (map2 (λ S1 S2. (Pp-i S1) ∪ (Rp-i (S2 ∪ b))) LS1 LS2) =*
$(\bigcap\ s \in \{y\ .\ y < length\ (map2\ (\lambda\ S1\ S2.\ (Pp\text{-}i\ (S1)\ ) \cup (Rp\text{-}i\ (\ S2 \cup\ b)))\ LS1\ LS2)\}.$
(map2 (λ S1 S2. (Pp-i (S1)) ∪ (Rp-i (S2 ∪ b))) LS1 LS2)!s)
by (*metis inter-list-defs*)
have *2*: *length (map2 (λ S1 S2. (Pp-i (S1)) ∪ (Rp-i (S2 ∪ b))) LS1 LS2) = length LS1*
using *assms* **by** *simp*
have *3*: $(\bigcap\ s \in \{y\ .\ y < length\ (map2\ (\lambda\ S1\ S2.\ (Pp\text{-}i\ (S1)) \cup (Rp\text{-}i\ (\ S2 \cup\ b)))\ LS1\ LS2)\}.$
(map2 (λ S1 S2. (Pp-i (S1)) ∪ (Rp-i (S2 ∪ b))) LS1 LS2)!s) =
$(\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (\ (LS2!s) \cup\ b)))$
using *assms* **by** *auto*
show *?thesis*
using *1 3* **by** *argo*
qed

lemma *inter-list-add-Pp-i*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = length LS2
finitary a
shows *(inter-list (map2 (λ S1 S2. ((Pp-i S1)) ∪ ((Rp-i S2))) LS1 LS2) ∪ Pp-i a) =*
inter-list (map2 (λ S1 S2. ((Pp-i S1) ∪ ((Rp-i S2)))) (map (λ S. S ∪ (Fin − (((S $\mathcal{S}$ (Fin − a)) · Empty)))) LS1) LS2)
proof −
have *1*: *(inter-list (map2 (λ S1 S2. ((Pp-i S1)) ∪ ((Rp-i S2))) LS1 LS2) ∪ Pp-i a) =*
$((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s))) \cup Pp\text{-}i\ a)$

using *assms reactivity-defs-alt[of LS1 LS2]* **by** *argo*
have *2*: $((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s))) \cup Pp\text{-}i\ a) =$
$((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s)) \cup (Pp\text{-}i\ a)\ )\ )$
by *blast*
have *31*: $\bigwedge\ s.\ s \in \{y\ .\ y < length\ LS1\} \Longrightarrow finitary\ (LS1!s)$
using *assms finitary-list-defs* **by** *blast*
have *3*: $((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s)) \cup (Pp\text{-}i\ a)\ )\ ) =$
$((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s \cup (Fin - (((LS1!s\ \mathcal{S}\ (Fin - a)) \cdot Empty))))) \cup (Rp\text{-}i\ (LS2!s))\ \ )\ )$
using *Pp-i-union[of - a] 31 assms(4)* **by** *blast*

have *4*: $((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s \cup (Fin - (((LS1!s\ \mathcal{S}\ (Fin - a)) \cdot Empty))))) \cup (Rp\text{-}i\ (LS2!s))\ \ )) =$
$inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ (S1 \cup (Fin - (((S1\ \mathcal{S}\ (Fin - a)) \cdot Empty))))\ )\ \cup (\ (Rp\text{-}i\ S2)))))\ LS1\ LS2)$
using *reactivity-defs-alt-add-Pp-i*[*of LS1 LS2 a*]
using *assms* **by** *blast*
have *5*: $inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ (S1 \cup (Fin - (((S1\ \mathcal{S}\ (Fin - a)) \cdot Empty))))\ )\ \cup (\ (Rp\text{-}i\ S2)))))\ LS1\ LS2) =$
$inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ S1\ )\ \cup (\ (Rp\text{-}i\ S2))))\ (map\ (\lambda\ S.\ S \cup (Fin - (((S\ \mathcal{S}\ (Fin - a)) \cdot Empty))))\ LS1)\ \ LS2)$
by (*simp add*: *map-zip-map*)
show *?thesis*
using *1 2 3 4 5* **by** *presburger*
qed

lemma *inter-list-add-Rp-i*:
assumes *finitary-list LS1*
finitary-list LS2
$length\ LS1 = length\ LS2$
finitary b
shows $(inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ S1))\ \cup (\ (Rp\text{-}i\ S2)))\ LS1\ LS2) \cup Rp\text{-}i\ b) =$
$inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ S1\ )\ \cup (\ (Rp\text{-}i\ S2))))\ LS1\ \ (map\ (\lambda S.\ S \cup b)\ LS2))$
proof −
have *1*: $(inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ S1))\ \cup (\ (Rp\text{-}i\ S2)))\ LS1\ LS2) \cup Rp\text{-}i\ b) =$
$((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s))) \cup Rp\text{-}i\ b)$
using *assms reactivity-defs-alt*[*of LS1 LS2*] **by** *argo*
have *2*: $((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s))) \cup Rp\text{-}i\ b) =$
$((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s)) \cup (Rp\text{-}i\ b)\ )\ )$
by *blast*
have *31*: $\bigwedge s.\ s \in \{y\ .\ y < length\ LS1\} \Longrightarrow finitary\ (LS2!s)$
using *assms* **by** (*simp add*: *finitary-list-defs*)
have *3*: $((\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (LS2!s)) \cup (Rp\text{-}i\ b)\ )\ ) =$
$(\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ ((LS2!s) \cup\ (\ b))\ )\ )$
using *Rp-i-union*[*of - b*] *31 assms* **by** *blast*
have *4*: $(\bigcap\ s \in \{y\ .\ y < length\ LS1\}.\ (Pp\text{-}i\ (LS1!s)) \cup (Rp\text{-}i\ (\ (LS2!s) \cup\ (\ b))\ )\ ) =$
$inter\text{-}list\ (map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ S1\ )\ \cup Rp\text{-}i\ (\ S2 \cup\ \ b)))\ LS1\ LS2)$
using *assms reactivity-defs-alt-add-Rp-i*[*of LS1 LS2 b*] **by** *simp*
have *5*: $(map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ S1\ )\ \cup Rp\text{-}i\ (\ S2 \cup\ \ b)))\ LS1\ LS2) =$
$(map2\ (\lambda\ S1\ S2.\ ((Pp\text{-}i\ S1\ )\ \cup (\ (Rp\text{-}i\ S2))))\ LS1\ \ (map\ (\lambda S.\ \ S \cup\ \ b)\ LS2))$
by (*simp add*: *map-equality-iff*)
show *?thesis*
using *1 2 3 4 5* **by** *argo*
qed

lemma *inter-list-add-Pp-i-Rp-i*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = length LS2
finitary a
finitary b
shows *inter-list (map2 (λ S1 S2. ((Pp-i S1)) ∪ ((Rp-i S2))) LS1 LS2) ∪ (Pp-i a ∩ Rp-i b) =*
(inter-list (map2 (λ S1 S2. ((Pp-i S1) ∪ ((Rp-i S2)))) (map (λ S. S ∪ (Fin − (((S 𝒮 (Fin − a)) · Empty)))) LS1) LS2) ∩
inter-list (map2 (λ S1 S2. ((Pp-i S1) ∪ ((Rp-i S2)))) LS1 (map (λS. S ∪ b) LS2)))
using *assms inter-list-add-Pp-i inter-list-add-Rp-i*
by *blast*

lemma *inter-list-add-Pp-i-Rp-i-cat*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = length LS2
finitary a
finitary b
shows *(inter-list (map2 (λ S1 S2. ((Pp-i S1) ∪ ((Rp-i S2)))) (map (λ S. S ∪ (Fin − (((S 𝒮 (Fin − a)) · Empty)))) LS1) LS2) ∩*
inter-list (map2 (λ S1 S2. ((Pp-i S1) ∪ ((Rp-i S2)))) LS1 (map (λS. S ∪ b) LS2))) =
inter-list (map2 (λ S1 S2. ((Pp-i S1) ∪ ((Rp-i S2))))
((map (λ S. S ∪ (Fin − (((S 𝒮 (Fin − a)) · Empty)))) LS1)@LS1) (LS2@(map (λS. S ∪ b) LS2)))
using *assms*
by (*simp add*: *inter-list-append*)

lemma *inter-list-add-Pp-i-Rp-i-add-left-right*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = length LS2
finitary a
finitary b
shows *inter-list (map2 (λ S1 S2. ((Pp-i S1)) ∪ ((Rp-i S2))) LS1 LS2) ∪ (Pp-i a ∩ Rp-i b) =*
inter-list (map2 (λ S1 S2. ((Pp-i S1) ∪ ((Rp-i S2)))) (add-left-p a LS1) (add-right-r b LS2))
using *assms*
by (*simp add*: *add-left-p-def add-right-r-def inter-list-add-Pp-i-Rp-i inter-list-add-Pp-i-Rp-i-cat*)

lemma *length-add-left-p*:
shows *length (add-left-p a LS1) = 2∗length LS1*
unfolding *add-left-p-def* **by** *auto*

lemma *length-add-rigth-r*:
shows *length* (*add-right-r a LS1*) = *2*∗*length LS1*
unfolding *add-right-r-def* **by** *auto*

lemma *length-add-right-r-list-Nil*:
length (*add-right-r-list* [] *LS1*) = *length LS1*
by *simp*

lemma *length-add-right-r-list-append*:
shows *length* (*add-right-r-list LS2* (*LS1* @ *LS3*)) =
length (*add-right-r-list LS2 LS1*) +
length (*add-right-r-list LS2 LS3*)
proof (*induction LS2 arbitrary*: *LS1 LS3*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **by** (*simp add*: *add-right-r-def*)
qed

lemma *length-add-right-r-list-equal*:
assumes *length LS1* = *length LS3*
shows *length* (*add-right-r-list LS2 LS3*) = *length* (*add-right-r-list LS2 LS1*)
using *assms*
proof (*induction LS2 arbitrary*: *LS1 LS3*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **by** (*simp add*: *length-add-rigth-r*)
qed

lemma *length-add-right-r-list-Cons*:
length (*add-right-r-list* (*a* # *LS2*) *LS1*) = *2*∗(*length* (*add-right-r-list LS2 LS1*))
proof −
have *1*: *length* (*add-right-r-list* (*a* # *LS2*) *LS1*) = *length* (*add-right-r-list LS2* (*add-right-r a LS1*))
by *simp*
have *2*: *length* (*add-right-r-list LS2* (*add-right-r a LS1*)) =
length (*add-right-r-list LS2* (*LS1* @ *map* (λ*S*. *S* ∪ *a*) *LS1*))
unfolding *add-right-r-def* **by** *simp*
have *3*: *length* (*add-right-r-list LS2* (*LS1* @ *map* (λ*S*. *S* ∪ *a*) *LS1*)) =
length (*add-right-r-list LS2 LS1*) +
length (*add-right-r-list LS2* (*map* (λ*S*. *S* ∪ *a*) *LS1*))
by (*simp add*: *length-add-right-r-list-append*)
have *4*: *length* (*add-right-r-list LS2* (*map* (λ*S*. *S* ∪ *a*) *LS1*)) =
length (*add-right-r-list LS2 LS1*)

```
  by (metis length-add-right-r-list-equal length-map)
 show ?thesis
 by (simp add: 2 3 4)
qed

lemma length-add-right-r-left-p-list:
 assumes length LS2 = length LS1
         length X = length Y
 shows length (add-right-r-list (map ((−) Fin) LS2) X) = length (add-left-p-list (map ((−) Fin) LS1) Y)
using assms
proof (induction LS2 LS1 arbitrary: X Y rule: list-induct2 )
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case
  proof −
   have 1: length (add-right-r-list (map ((−) Fin) (x # xs)) X) =
           length (add-right-r-list (map ((−) Fin) xs) (add-right-r (Fin − x) X))
     by simp
   have 2: length (add-left-p-list (map ((−) Fin) (y # ys)) Y) =
           length (add-left-p-list (map ((−) Fin) ys) (add-left-p (Fin − y) Y))
     by simp
   have 3: length (add-right-r-list (map ((−) Fin) xs) (add-right-r (Fin − x) X)) =
           length (add-left-p-list (map ((−) Fin) ys) (add-left-p (Fin − y) Y))
     using Cons.IH[of (add-right-r (Fin − x) X) (add-left-p (Fin − y) Y)]
       by (simp add: Cons.prems add-left-p-def add-right-r-def)
   show ?thesis
   by (simp add: 3)
 qed
qed

lemma length-add-right-r-list:
 (length (add-right-r-list LS2 LS1)) = 2^(length LS2) ∗ length LS1
proof (induction LS2 arbitrary: LS1)
case Nil
then show ?case by simp
next
case (Cons a LS2)
then show ?case
by (metis length-Cons length-add-right-r-list-Cons mult.assoc  power-Suc)
qed


lemma finitary-list-map-union-p:
 assumes finitary-list LS1
         finitary x
 shows  finitary-list (map (λS. S ∪ (Fin − (((S 𝒮 (Fin − x)) · Empty)))) LS1)
using assms
proof (induction LS1)
```

case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS1*)
then show *?case*
proof −
have *1*: *finitary a*
using *Cons.prems*(*1*) *finitary-list-defs* **by** *force*
have *2*: *finitary x*
by (*simp add*: *assms*(*2*))
have *3*: *finitary* (*a* ∪ (*Fin* − (((*a* $\mathcal{S}$ (*Fin* − *x*)) · *Empty*))))
using *1 subseteq-Fin-eq-finitary* **by** *auto*
have *4*: *finitary-list* (*map* (λ*S*. *S* ∪ (*Fin* − (((*S* $\mathcal{S}$ (*Fin* − *x*)) · *Empty*)))) (*a* # *LS1*)) =
finitary (*a* ∪ (*Fin* − (((*a* $\mathcal{S}$ (*Fin* − *x*)) · *Empty*)))) ∧ *finitary-list* (*map* (λ*S*. *S* ∪ (*Fin* − (((*S* $\mathcal{S}$ (*Fin* − *x*)) · *Empty*)))) *LS1*)
unfolding *finitary-list-def* **apply** *simp*
by (*metis* (*lifting*) *ext 2 Cons.IH Cons.prems*(*1*) *finitary-list-append finitary-list-def rev-append rev-eq-Cons-iff rev-rev-ident*)
show *?thesis*
using *3 4* **by** *auto*
qed
qed

lemma *finitary-list-add-left-p*:
assumes *finitary-list LS1*
finitary a
shows *finitary-list* (*add-left-p a LS1*)
using *assms* **unfolding** *add-left-p-def*
by (*simp add*: *finitary-list-append finitary-list-map-union-p*)

lemma *finitary-list-add-right-r*:
assumes *finitary-list LS2*
finitary b
shows *finitary-list* (*add-right-r b LS2*)
using *assms* **unfolding** *add-right-r-def*
by (*simp add*: *finitary-list-append finitary-list-map-union*)

lemma *inter-list-union-list-rp-absorb*:
assumes *finitary-list LS1*
finitary-list LS2
length LS1 = *length LS2*
finitary-list LS3
finitary-list LS4
length LS3 = *length LS4*
shows *inter-list* (*map2* (λ *S1 S2*. ((*Pp-i S1*)) ∪ ((*Rp-i S2*))) *LS1 LS2*) ∪
union-list (*map2* (λ *S1 S2*. ((*Pp-i S1*) ∩ ((*Rp-i S2*)))) *LS3 LS4*) =
inter-list (*map2* (λ *S1 S2*. ((*Pp-i S1*)) ∪ ((*Rp-i S2*)))
(*add-left-p-list LS3 LS1*) (*add-right-r-list LS4 LS2*))

using *assms*
proof (*induction LS3 arbitrary*: *LS4 LS1 LS2*)
case *Nil*
then show *?case* **unfolding** *union-list-def* **by** *simp*
next
case (*Cons a LS3a*)
then show *?case*
proof −
have *1*: ∃ *b LS4b*. *LS4* = *b* # *LS4b*
by (*metis Cons.prems*(*6*) *length-Suc-conv*)
obtain *b LS4b* **where** *2*: *LS4* = *b* # *LS4b*
using *1* **by** *blast*
have *3*: *union-list* (*map2* (λ *S1 S2*. ((*Pp-i S1*) ∩ ((*Rp-i S2*)))) (*a*#*LS3a*) *LS4*) =
((*Pp-i a*) ∩ ((*Rp-i b*))) ∪ *union-list* (*map2* (λ *S1 S2*. ((*Pp-i S1*) ∩ ((*Rp-i S2*)))) *LS3a LS4b*)
using *2* **unfolding** *union-list-def* **by** *simp*
have *4*: *finitary a*
using *Cons.prems*(*4*) *finitary-list-defs* **by** *force*
have *5*: *finitary b*
using *2 Cons.prems*(*5*) *finitary-list-defs* **by** *fastforce*
have *6*: *inter-list* (*map2* (λ *S1 S2*. ((*Pp-i S1*)) ∪ ((*Rp-i S2*))) *LS1 LS2*) ∪ ((*Pp-i a*) ∩ ((*Rp-i b*))) =
inter-list (*map2* (λ *S1 S2*. ((*Pp-i S1*)) ∪ ((*Rp-i S2*))) (*add-left-p a LS1*) (*add-right-r b LS2*))
using *inter-list-add-Pp-i-Rp-i-add-left-right*[*of LS1 LS2 a b*]
using *4 5 Cons.prems*(*1*,*2*,*3*) **by** *linarith*
have *7*: *inter-list* (*map2* (λ*x y*. *Pp-i x* ∪ *Rp-i y*) (*add-left-p-list* (*a* # *LS3a*) *LS1*) (*add-right-r-list* (*b* # *LS4*) *LS2*)) =
inter-list (*map2* (λ*x y*. *Pp-i x* ∪ *Rp-i y*) ((*add-left-p-list LS3a* (*add-left-p a LS1*))) ((*add-right-r-list LS4* (*add-right-r b LS2*))))
by *simp*
have *8*: *finitary-list LS3a*
by (*metis Cons.prems*(*4*) *Cons-eq-append-conv finitary-list-append self-append-conv2*)
have *9*: *finitary-list LS4b*
by (*metis 2 Cons.prems*(*5*) *Suc-less-eq2 finitary-list-defs length-Cons nth-Cons-Suc*)
have *10*: *finitary-list* (*add-left-p a LS1*)
unfolding *add-left-p-def finitary-list-def* **using** *Cons.prems*
by (*metis 4 add-left-p-def finitary-list-add-left-p finitary-list-def*)
have *11*: *finitary-list* (*add-right-r b LS2*)
by (*simp add*: *5 Cons.prems*(*2*) *finitary-list-add-right-r*)
have *12*: *inter-list* (*map2* (λ*x y*. *Pp-i x* ∪ *Rp-i y*) (*add-left-p a LS1*) (*add-right-r b LS2*)) ∪
union-list (*map2* (λ*x y*. *Pp-i x* ∩ *Rp-i y*) *LS3a LS4b*) =
inter-list (*map2* (λ*x y*. *Pp-i x* ∪ *Rp-i y*) (*add-left-p-list LS3a* (*add-left-p a LS1*))
(*add-right-r-list LS4b* (*add-right-r b LS2*)))
using *Cons.IH*[*of add-left-p a LS1 add-right-r b LS2 LS4b*]
using *2 8 9 Cons.prems*(*1*,*2*,*3*,*6*)
by (*simp add*: *10 11 length-add-left-p length-add-rigth-r*)
show *?thesis*
using *12 2 3 6* **by** *auto*
qed
qed

lemma *finitary-list-add-right-r-list*:
assumes *finitary-list LS2*
finitary-list X
shows *finitary-list* (*add-right-r-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) *X*)
using *assms*
proof (*induction LS2 arbitrary*: *X*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **apply** *simp*
by (*metis add-right-r-def append-Cons append-Nil finitary-list-append finitary-list-map-union finitary-Fin-complement*
qed

lemma *finitary-list-add-left-p-list*:
assumes *finitary-list LS1*
finitary-list X
shows *finitary-list* (*add-left-p-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) *X*)
using *assms*
proof (*induction LS1 arbitrary*: *X*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS1*)
then show *?case* **apply** *simp*
using *finitary-list-add-left-p finitary-list-def finitary-Fin-complement* **by** *fastforce*
qed

lemma *length-add-right-r-list-gr-zer0*:
assumes *length LS2* > *0*
length X > *0*
shows *length* (*add-right-r-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) *X*) >*0*
using *assms*
proof (*induction LS2 arbitrary*: *X*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a LS2*)
then show *?case* **apply** *simp*
by (*metis add-right-r-def add-right-r-list.simps*(*1*) *append-is-Nil-conv list.map-disc-iff*)
qed

lemma *complement-reactivity-prop*:
assumes *reactivity-prop S k*
shows *reactivity-prop* (*Inf* −*S*) (*2*^*k*)
proof −
obtain *LS1 LS2* **where** *1*: *finitary-list LS1* ∧ *finitary-list LS2* ∧

length LS1 = length LS2 ∧ length LS1 >0 ∧ length LS1 = k ∧
S = inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2)
using *assms reactivity-prop-def* **by** *blast*
have *2*: *Inf − S = Inf − (inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2))*
using *1 assms* **by** *blast*
have *3*: *Inf − (inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) LS1 LS2)) =*
union-list (map2 (λ S1 S2. Inf − ((Rp-i S1) ∪ (Pp-i S2))) (LS1) (LS2))
using *1 complement-inter-list-1-Rp-Pp* **by** *blast*
have *4*: *union-list (map2 (λ S1 S2. Inf − ((Rp-i S1) ∪ (Pp-i S2))) (LS1) (LS2)) =*
union-list (map2 (λ S1 S2. ((Pp-i S1)) ∩ ((Rp-i S2))) (map (λS. Fin −S) LS1) (map (λS. Fin−S) LS2))
by (*simp add*: *1 complement-inter-list-2-Rp-Pp*)
have *5*: *Fs = inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) [Fs] [Fs])*
unfolding *inter-list-def* **by** (*simp add*: *Rp-i-Fs Pp-i-Fs*)
have *6*: *union-list (map2 (λ S1 S2. ((Pp-i S1)) ∩ ((Rp-i S2))) (map (λS. Fin −S) LS1) (map (λS. Fin−S) LS2)) =*
inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) [Fs] [Fs]) ∪
union-list (map2 (λ S1 S2. ((Pp-i S1)) ∩ ((Rp-i S2))) (map (λS. Fin −S) LS1) (map (λS. Fin−S) LS2))
using *5 Fs-def* **by** *auto*
have *7*: *finitary-list [Fs]*
by (*simp add*: *finitary-Fs finitary-list-defs*)
have *8*: *finitary-list (map ((−) Fin) LS1)*
by (*simp add*: *Fin-def finitary-def finitary-list-defs*)
have *9*: *finitary-list (map ((−) Fin) LS2)*
by (*simp add*: *Fin-def finitary-def finitary-list-defs*)
have *10*: *inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2)) [Fs] [Fs]) ∪*
union-list (map2 (λ S1 S2. ((Pp-i S1)) ∩ ((Rp-i S2))) (map (λS. Fin −S) LS1) (map (λS. Fin−S) LS2)) =
inter-list (map2 (λ S1 S2. ((Pp-i S1)) ∪ ((Rp-i S2)))
(add-left-p-list (map (λS. Fin −S) LS1) [Fs]) (add-right-r-list (map (λS. Fin−S) LS2) [Fs]))
using *inter-list-union-list-rp-absorb*[*of* [*Fs*] [*Fs*] (*map* (λ*S*. *Fin* −*S*) *LS1*) (*map* (λ*S*. *Fin*−*S*) *LS2*)]
by (*metis 1 7 8 9 length-map reactivity-defs-swap*)
have *11*: *length (add-right-r-list (map ((−) Fin) LS2) [Fs]) = length (add-left-p-list (map ((−) Fin) LS1) [Fs])*
by (*simp add*: *1 length-add-right-r-left-p-list*)
have *12*: *inter-list (map2 (λ S1 S2. ((Pp-i S1)) ∪ ((Rp-i S2)))*
(add-left-p-list (map (λS. Fin −S) LS1) [Fs]) (add-right-r-list (map (λS. Fin−S) LS2) [Fs])) =
inter-list (map2 (λ S1 S2. (Rp-i S1) ∪ (Pp-i S2))
(add-right-r-list (map (λS. Fin−S) LS2) [Fs]) (add-left-p-list (map (λS. Fin −S) LS1) [Fs]))
using *reactivity-defs-swap*[*of* (*add-right-r-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*])
(*add-left-p-list* (*map* (λ*S*. *Fin* −*S*) *LS1*) [*Fs*])]
using *11* **by** *blast*
have *13*: *finitary-list (add-right-r-list (map (λS. Fin−S) LS2) [Fs])*
by (*simp add*: *1 7 finitary-list-add-right-r-list*)
have *14*: *finitary-list (add-left-p-list (map (λS. Fin −S) LS1) [Fs])*
by (*simp add*: *1 7 finitary-list-add-left-p-list*)
have *15*: *length (add-right-r-list (map (λS. Fin−S) LS2) [Fs]) >0*
by (*metis 1 length-Cons length-add-right-r-list-gr-zer0*
zero-less-Suc)

have *16*: *length* (*add-right-r-list* (*map* (λ*S*. *Fin*−*S*) *LS2*) [*Fs*]) = *2*^*k*
by (*metis* (*no-types*, *opaque-lifting*) *1 length-Cons length-add-right-r-list length-map list.size*(*3*)
nat-power-eq-Suc-0-iff power-Suc power-Suc0-right)
show *?thesis* **unfolding** *reactivity-prop-def*
by (*metis 10 11 12 13 14 15 16 2 3 4 6 Diff-subset subseteq-Inf-eq-infinitary*)
qed

lemma *complement-reactivity-prop-a*:
assumes *S* ⊆ *Inf*
reactivity-prop (*Inf* −*S*) *k*
shows *reactivity-prop S* (*2*^*k*)
using *assms*
by (*metis Diff-Diff-Int complement-reactivity-prop inf.absorb-iff2*)

lemma *reactivity-prop-inter*:
assumes *reactivity-prop S1 k*
reactivity-prop S2 l
shows *reactivity-prop* (*S1* ∩ *S2*) (*k*+*l*)
proof −
have *1*: (∃ *LS1 LS2*. *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *0* < *length LS1* ∧ *length LS1* = *k* ∧
S1 = *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS1 LS2*))
using *assms reactivity-prop-def*[*of S1*] **by** *blast*
obtain *LS1 LS2* **where** *2*: *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *0* < *length LS1* ∧ *length LS1*=*k* ∧
S1 = *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS1 LS2*)
using *1* **by** *blast*
have *3*: (∃ *LS1 LS2*. *finitary-list LS1* ∧ *finitary-list LS2* ∧
length LS1 = *length LS2* ∧ *0* < *length LS1* ∧ *length LS1*=*l*∧
S2 = *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS1 LS2*))
using *assms reactivity-prop-def* **by** *blast*
obtain *LS3 LS4* **where** *4*: *finitary-list LS3* ∧ *finitary-list LS4* ∧
length LS3 = *length LS4* ∧ *0* < *length LS3* ∧ *length LS3*=*l* ∧
S2 = *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS3 LS4*)
using *3* **by** *blast*
have *5*: *S1* ∩ *S2* = *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS1 LS2*) ∩
inter-list (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS3 LS4*)
using *2 4* **by** *blast*
have *6*: *inter-list* (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS1 LS2*) ∩
inter-list (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) *LS3 LS4*) =
inter-list (*map2* (λ *S1 S2*. (*Rp-i S1*) ∪ (*Pp-i S2*)) (*LS1*@*LS3*) (*LS2*@*LS4*))
by (*simp add*: *2 inter-list-append*)
have *7*: *finitary-list* (*LS1*@*LS3*)
by (*simp add*: *2 4 finitary-list-append*)
have *8*: *finitary-list* (*LS2*@*LS4*)
by (*simp add*: *2 4 finitary-list-append*)
have *9*: *length* (*LS1*@*LS3*) = *length* (*LS2*@*LS4*)
using *2 4* **by** *auto*

have *10*: *0 < length (LS1@LS3)*
using *2 4* **by** *blast*
have *11*: *length (LS1@LS3) = k+l*
using *2 4* **by** *fastforce*
show *?thesis* **unfolding** *reactivity-prop-def*
using *2 4 6 7 8 9 11*
by (*metis 10 assms(1) le-infI1 reactivity-prop-def subseteq-Inf-eq-infinitary*)
qed

lemma *reactivity-prop-union*:
assumes
reactivity-prop S1 k
reactivity-prop S2 l
shows *reactivity-prop (S1 ∪ S2) (2^((2^k) + (2^l)))*
proof −
have *1*: *reactivity-prop (Inf −S1) (2^k)*
by (*simp add*: *assms(1) complement-reactivity-prop*)
have *2*: *reactivity-prop (Inf −S2) (2^l)*
by (*simp add*: *assms(2) complement-reactivity-prop*)
have *3*: *reactivity-prop ((Inf −S1) ∩ (Inf −S2)) ((2^k) + (2^l))*
by (*simp add*: *1 2 reactivity-prop-inter*)
have *4*: *reactivity-prop (Inf −((Inf −S1) ∩ (Inf −S2))) (2^((2^k) + (2^l)))*
using *3 complement-reactivity-prop* **by** *blast*
have *5*: *(Inf −((Inf −S1) ∩ (Inf −S2))) = (S1 ∪ S2)*
by (*metis Diff-Diff-Int Diff-Int Diff-partition Un-Int-eq(1) assms(1,2) reactivity-prop-def subseteq-Inf-eq-infinitary*)
show *?thesis*
using *4 5* **by** *auto*
qed

lemma *response-is-reactivity*:
assumes *response-prop S*
shows *reactivity-prop S 1*
proof −
obtain *S1* **where** *1*: *finitary S1 ∧ infinitary S ∧ S = Rp-i S1*
using *assms response-prop-def* **by** *blast*
have *2*: *S = Rp-i S1 ∪ Pp-i Fs*
using *1 Fs-def Pp-i-Fs* **by** *fastforce*
have *3*: *finitary-list [S1]*
by (*simp add*: *1 finitary-list-defs*)
have *4*: *finitary-list [Fs]*
by (*simp add*: *finitary-Fs finitary-list-defs*)
have *5*: *length [S1] = length [Fs]*
by *simp*
have *6*: *0 < length [S1]*
by *simp*
have *7*: *length [S1] = 1*
by *simp*
have *8*: *S = inter-list (map2 (λx y. Rp-i x ∪ Pp-i y) [S1] [Fs])*

unfolding *inter-list-def* **using** *2* **by** *auto*
show *?thesis* **unfolding** *reactivity-prop-def*
using *1 3 4 5 7 8* **by** *blast*
qed

lemma *persistence-is-reactivity*:
assumes *persistence-prop S*
shows *reactivity-prop S 1*
proof −
obtain *S1* **where** *1*: *finitary S1* ∧ *infinitary S* ∧ *S* = *Pp-i S1*
using *assms persistence-prop-def* **by** *blast*
have *2*: *S* = *Rp-i Fs* ∪ *Pp-i S1*
using *1 Fs-def Rp-i-Fs* **by** *auto*
have *3*: *finitary-list* [*S1*]
by (*simp add*: *1 finitary-list-defs*)
have *4*: *finitary-list* [*Fs*]
by (*simp add*: *finitary-Fs finitary-list-defs*)
have *5*: *length* [*Fs*] = *length* [*S1*]
by *simp*
have *6*: *0* < *length* [*Fs*]
by *simp*
have *7*: *length* [*Fs*] = *1*
by *simp*
have *8*: *S* = *inter-list* (*map2* (λ*x y*. *Rp-i x* ∪ *Pp-i y*) [*Fs*] [*S1*])
unfolding *inter-list-def* **using** *2* **by** *auto*
show *?thesis* **unfolding** *reactivity-prop-def*
using *1 3 4 5 7 8* **by** *blast*
qed

7.8.11 Safety and liveness

definition *as-safety-prop*
where *as-safety-prop S* =
(*S* ⊆ *Inf* ∧
(∀ *x*. ¬*nfinite x* ⟶
(*x* ∉ *S* ⟷ (∃ *y*. *y* ≼ *x* ∧ (∀ *z*. ¬*nfinite z* ⟶ *nappend y z* ∉ *S*)))))

definition *as-liveness-prop*
where *as-liveness-prop S* =
(*S* ⊆ *Inf* ∧
(∀ *x*. *nfinite x* ⟶ (∃ *y*. ¬*nfinite y* ∧ *nappend x y* ∈ *S*)))

definition *liveness-prop*
where *liveness-prop S* = (*S* = (*S* ∪ (*Ep-i* (*Fin* − (*Pref S*)))))

lemma *safety-subset-eq*:
assumes *S* ⊆ *Inf*
shows *S* ⊆ *Ap-i*(*Pref S*)
proof −

have *1*: $\bigwedge x$. $x \in S \cap$ *Ap-i* (*Pref S*) $\longleftrightarrow x \in S$
using *assms* **unfolding** *Ap-def Ap-i-def Inf-def Pref-def* **by** *auto*
show *?thesis*
using *1* **by** *auto*
qed

lemma *eq-Ap-i-Pref*:
assumes $S \subseteq$ *Inf*
shows $S =$ *Ap-i* (*Pref S*) $\longleftrightarrow$
$(\forall\ x.\ \ x \in S = (\neg$*nfinite* $x \wedge (\forall\ \sigma 1.\ \sigma 1 \preceq x \longrightarrow \sigma 1 \in$ *Pref S*$)))$
proof −
have *1*: $\bigwedge x.$ $(x \in$ *Ap-i* (*Pref S*)$) \longleftrightarrow$
$\neg$*nfinite* $x \wedge (\forall\ \sigma 1.\ \sigma 1 \preceq x \longrightarrow \sigma 1 \in$ *Pref S*$)$
using *assms* **unfolding** *Ap-i-def Ap-def Inf-def Pref-def* **by** *auto*
show *?thesis*
using *set-eq-iff*[*of S Ap-i* (*Pref S*)] *assms 1* **by** *presburger*
qed

lemma *safety-prop-0*:
assumes $S \subseteq$ *Inf*
shows *safety-prop* $S \longleftrightarrow$
$(\forall\ x.\ \ x \in S = (\neg$*nfinite* $x \wedge (\forall\ \sigma 1.\ \sigma 1 \preceq x \longrightarrow \sigma 1 \in$ *Pref S*$)))$
proof −
have *31*: *safety-prop* $S \longleftrightarrow (\exists$ *S1*. *finitary S1* $\wedge$ *infinitary* $S \wedge S =$ *Ap-i S1*$)$
unfolding *safety-prop-def* **by** *blast*
have *32*: *infinitary* (*Ap-i* (*Pref S*))
by (*metis Ap-i-inter-Inf inf-commute inf-le1 subseteq-Inf-eq-infinitary*)
have *33*: *finitary* (*Pref S*)
by (*simp add*: *finitary-Pref*)
have *34*: $S =$ *Ap-i* (*Pref S*) $\Longrightarrow$ *safety-prop S*
unfolding *safety-prop-def* **using** *32 33* **by** *auto*
have *35*: *safety-prop* $S \Longrightarrow (\forall\ x.\ \ x \in S = (\neg$*nfinite* $x \wedge (\forall\ \sigma 1.\ \sigma 1 \preceq x \longrightarrow \sigma 1 \in$ *Pref S*$)))$
unfolding *safety-prop-def Ap-i-def Int-iff Pref-def Inf-def*
using *in-Ap mem-Collect-eq nprefix-ntaken-infinite order-refl subsetD subseteq-Fin-eq-finitary*
by *fastforce*
show *?thesis*
using *34 35 assms eq-Ap-i-Pref* **by** *auto*
qed

lemma *safety-prop-1*:
assumes $S \subseteq$ *Inf*
shows *safety-prop* $S \longleftrightarrow S =$ *Ap-i*(*Pref S*)
by (*simp add*: *assms eq-Ap-i-Pref safety-prop-0*)

lemma *safety-eq-as-safety*:
assumes $S \subseteq$ *Inf*
shows *safety-prop* $S \longleftrightarrow$ *as-safety-prop S*

proof −
have *1*: *safety-prop S* ⟹ *as-safety-prop S*
using *assms* **by** (*metis as-safety-prop-def nfinite-nappend nprefix-def nprefix-eq-def safety-prop-0*)
have *2*: *as-safety-prop S* ⟹ *safety-prop S*
unfolding *as-safety-prop-def* **using** *assms safety-prop-0*[*of S*]
apply (*simp add*: *Pref-def*)
by (*meson infinitary-def subseteq-Inf-eq-infinitary*)
show *?thesis* **using** *1 2* **by** *auto*
qed

lemma *eq-Ep-i-not-Pref*:
assumes *S* ⊆ *Inf*
shows (*S* = (*S* ∪ (*Ep-i* (*Fin* − (*Pref S*))))) ⟷
(∀ *x*. *x*∈ *S* =
(¬ *nfinite x* ∧
(*x* ∈ *S* ∨
(∃ σ*1*. σ*1* ≼ *x* ∧ (∀ σ*1a*. σ*1* ≼ σ*1a* ⟶ σ*1a* ∉ *S*)))))
proof −
have *1*: ⋀*x*. (*x* ∈ (*S* ∪ (*Ep-i* (*Fin* − (*Pref S*))))) ⟷
¬ *nfinite x* ∧ (*x* ∈ *S* ∨ (∃ σ*1*. σ*1* ≼ *x* ∧ (∀ σ*1a*. σ*1* ≼ σ*1a* ⟶ σ*1a* ∉ *S*)))
using *assms* **unfolding** *Ep-i-def Ep-def Inf-def Fin-def Pref-def infinitary-def*
apply *simp*
using *nprefix-eq-exists-ntaken* **by** *blast*
show *?thesis* **using** *1 set-eq-iff*[*of S* (*S* ∪ (*Ep-i* (*Fin* − (*Pref S*))))]
by *presburger*
qed

lemma *liveness-0*:
assumes *S* ⊆ *Inf*
shows *liveness-prop S* ⟷
(∀ *x*. *x*∈ *S* =
(¬ *nfinite x* ∧
(*x* ∈ *S* ∨
(∃ σ*1*. σ*1* ≼ *x* ∧ (∀ σ*1a*. σ*1* ≼ σ*1a* ⟶ σ*1a* ∉ *S*)))))
unfolding *liveness-prop-def* **using** *assms eq-Ep-i-not-Pref*[*of S*] **by** *auto*

lemma *liveness-1*:
assumes *S* ⊆ *Inf*
shows *liveness-prop S* ⟷ (∀ *x*. ¬*nfinite x* ⟶ *x* ∈ *Ap-i*(*Pref S*))
proof −
have *1*: *liveness-prop S* ⟷ (*S* = *S* ∪ *Ep-i* (*Fin* − *Pref S*))
unfolding *liveness-prop-def* **by** *auto*
have *2*: *Ep-i* (*Fin* − *Pref S*) = *Inf* − *Ap-i* (*Pref S*)
by (*simp add*: *Ap-i-eqv-Ep-i finitary-Pref subseteq-Fin-eq-finitary*)
have *3*: *S* ∪ *Ep-i* (*Fin* − *Pref S*) = *S* ∪ (*Inf* − *Ap-i* (*Pref S*))
using *2* **by** *auto*
have *4*: *S* = (*Inf* − (*Inf* − *S*))
by (*simp add*: *assms double-diff*)
have *5*: *S* ∪ (*Inf* − *Ap-i* (*Pref S*)) =

```
        (Inf − (Inf − S)) ∪ (Inf − Ap-i (Pref S))
 using 4 by auto
 have 6: (Inf − (Inf − S)) ∪ (Inf − Ap-i (Pref S)) =
        (Inf − ((Inf − S) ∩  Ap-i (Pref S)))
   by auto
 have 7:  Inf − (S ∪ Ep-i (Fin − Pref S)) =
         ((Inf − S) ∩ Ap-i(Pref S))
   using 3 by blast
 have 8: (S = S ∪ Ep-i (Fin − Pref S)) ⟷
        ((Inf − S) =  ((Inf − S) ∩ Ap-i (Pref S)))
   using  3 4 7 Diff-Int  by metis
 have 9: ((Inf − S) =  ((Inf − S) ∩ Ap-i (Pref S))) ⟷
        ((Inf −S) ⊆ Ap-i (Pref S))
   by auto
 have 10: liveness-prop S ⟷ ((Inf −S) ⊆ Ap-i (Pref S))
   by (simp add: 1 8 9)
 have 11:  S ⊆ Ap-i (Pref S)
  by (simp add: assms safety-subset-eq)
 have 12: ((Inf −S) ⊆ Ap-i (Pref S)) ⟷ (Inf −S) ∪ S ⊆ Ap-i (Pref S)
    using 11 by auto
 have 13: (Inf −S) ∪ S ⊆ Ap-i (Pref S) ⟷
         (∀ x. ¬nfinite x ⟶ x ∈ Ap-i(Pref S))
   unfolding Inf-def  using 11 by auto
 show ?thesis
 using 10 12 13 by blast
qed


lemma liveness-eq-as-liveness:
 assumes  S ⊆ Inf
 shows  liveness-prop S ⟷ as-liveness-prop S
proof −
 have 1:  liveness-prop S ⟹ as-liveness-prop S
  using assms
  by (metis as-liveness-prop-def liveness-0 nfinite-nappend nfinite-niterates nprefix-def nprefix-eq-def)
 have 2: as-liveness-prop S ⟹ liveness-prop S
   unfolding as-liveness-prop-def using assms liveness-0[of S]
    apply simp
   by (metis infinitary-def nprefix-eq-exists-ntaken nprefix-nappend subseteq-Inf-eq-infinitary)
 show ?thesis using 1 2 by auto
qed


lemma safety-liveness:
  assumes  S ⊆ Inf
 shows  S = (Ap-i (Pref S) ∩ (S ∪ (Ep-i (Fin − (Pref S)))))
proof −
 have 1: (Ep-i (Fin − (Pref S))) = (Inf − (Ap-i (Pref S)))
   by (simp add: Ap-i-eqv-Ep-i finitary-Pref  subseteq-Fin-eq-finitary)
 have 2: (Ap-i (Pref S) ∩ (S ∪ (Inf − (Ap-i (Pref S))))) = S ∩ Ap-i (Pref S)
```

by *blast*
have *4*: $S \cap$ *Ap-i* (*Pref S*) = *S*
using *safety-subset-eq assms* **by** *blast*
show *?thesis*
by (*simp add*: *1 2 4*)
qed

end

Bibliography

[1] A. Armstrong, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2013. https://isa-afp.org/entries/Kleene_Algebra.shtml, Formal proof development.

[2] H. Bowman and S. J. Thompson. A Decision Procedure and Complete Axiomatization of Finite Interval Temporal Logic with Projection. *Journal of Logic and Computation*, 13(2):195–239, Apr. 2003.

[3] A. Cau and B. Moszkowski. Fusion logic, 2026. http://antonio-cau.co.uk/ITL/itlhomepagesu18.html#flcheck-fusion-logic-decision-procedure.

[4] G. Grov and S. Merz. A Definitional Encoding of TLA* in Isabelle/HOL. *Archive of Formal Proofs*, 2011. https://www.isa-afp.org/entries/TLA.html, Formal proof development.

[5] D. P. Guelev and B. Moszkowski. A separation theorem for discrete-time interval temporal logic. *Journal of Applied Non-Classical Logics*, 32(1):28–54, 2022.

[6] A. Lochbihler. Coinductive. *Archive of Formal Proofs*, feb 2010. https://www.isa-afp.org/entries/Coinductive.html, Formal proof development.

[7] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 201–284. Springer, 1988.

[8] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.

[9] S. Merz. An Encoding of TLA in Isabelle. https://www.pst.informatik.uni-muenchen.de/~merz/isabelle/. Part of the Isabelle distribution., 1998.

[10] B. Moszkowski. Compositional reasoning about projected and infinite time. In *Proceedings of the First IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, pages 238–245. IEEE Computer Society Press, 1995. Download pdf.

[11] B. Moszkowski. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1–2):55–104, 2004.

[12] B. Moszkowski. Compositional reasoning using intervals and time reversal. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):175–250, 2014.

[13] B. Moszkowski and D. Guelev. An application of temporal projection to interleaving concurrency. *Formal Aspects of Computing*, 29(4):705–750, July 2017.

[14] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.

[15] B. C. Moszkowski. A Complete Axiom System for Propositional Interval Temporal Logic with Infinite Time. *Logical Methods in Computer Science Journal*, 8(3), 2012.

[16] B. C. Moszkowski and D. P. Guelev. An Application of Temporal Projection to Interleaving Concurrency. In *Dependable Software Engineering: Theories, Tools, and Applications - First International Symposium, SETTA 2015, Nanjing, China, November 4-6, 2015, Proceedings*, pages 153–167, 2015.

[17] Y. Venema. A modal logic for chopping intervals. *Journal of Logic and Computation*, 1(4):453–476, 09 1991.

[18] J. S. Warford, D. Vega, and S. M. Staley. A Calculational Deductive System for Linear Temporal Logic. https://www.cslab.pepperdine.edu/warford/Papers/Vega-Paper.pdf, 2019.